Technische Universität Berlin
Fakultät IV (Elektrotechnik und Informatik)
Institut für Telekommunikationssysteme
Kommunikations- und Betriebssysteme

SS 2007

Diplomarbeit

Entwicklung einer allgemeinen Teststrategie für zustandsbehaftete Paketfilter

cand. Inform. Nikolaus Filus

15. August 2007

Prof. Dr. Hans-Ulrich Heiß



Diese Arbeit wurde mit Hilfe von KOMA-Script und LaTeX gesetzt.

Network monitoring tells what has happened.
Network testing tells what will happen.

– The Art of Testing Network Systems
R.W.Buchanan, Wiley, 1996

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Das Internet hat sich von einem Forschungsnetzwerk zu einer weltweiten Kommunikationsinfrastruktur entwickelt, die zahlreiche kleinere Netzwerke miteinander verbindet. Verschiedene kommerzielle, freie, private, forschende, staatliche oder militärische Organisationen sind an das Internet angeschlossen, um Dienste wie das World Wide Web, Datentransfer sowie Kommunikationsmöglichkeiten wie E-Mail, Instant Messaging und neuerdings auch Telefonie zu nutzen. All diese Dienste basieren auf dem TCP/IP Protokollstapel, der in den 70er Jahren im Rahmen eines Projektes bei der Defense Advanced Research Projects Agency (DARPA) in Zusammenarbeit mit dem Massachusetts Institute of Technology (MIT) entwickelt wurde, um die aufkommenden Probleme und den Protokollwildwuchs im experimentellen ARPAnet Netzwerk zu beheben. Das verfolgte Ziel der Neuentwicklung war es eine einfache, effiziente und offene Infrastruktur für eine freundliche und kooperative Umgebung zu schaffen. Viele der späteren Einsatzszenarien waren damals noch nicht bekannt oder gar vorstellbar, weswegen die Entwürfe noch kaum Überlegungen zu Sicherheitsanforderungen wie Authentifizierung, Autorisierung, Zuordenbarkeit, Integrität oder Vertraulichkeit, die in der heutigen feindlichen Umgebung notwendig sind, beinhalteten. Für End-zu-End-Kommunikation wurden viele dieser Merkmale durch verschiedene Erweiterungen oder höhere Protokolle nachgerüstet. Um jedoch die Infrastruktur und ihre jeweiligen Teilnetze zu schützen oder den Zugang zu ihnen zu beschränken war es notwendig Schutzmaßnahmen einzurichten und den Zugriff zu regeln.

Die regelnden Komponenten, die eine Art (Brand-) Schutzmauer zwischen Netzwerken mit unterschiedlichen Sicherheitsanforderungen und Vertrauensstellungen darstellen, werden „Firewalls“ genannt und vereinen eine Reihe von verschiedenen Techniken und Mechanismen, die die Regeln durchsetzen.

Die ersten (kommerziellen) Firewalls erschienen 1989 und waren eine Kombination aus filternden Routern auf der Transportebene und Proxies auf Anwendungsebene (vgl. auch [IF02]). Sie entstanden aus der Erkenntnis, dass nicht alle Netzteilnehmer kooperative Ziele verfolgen, wozu sicherlich die ersten Netzwerkwürmer beigetragen haben. Aufgrund der zunehmenden Netzwerkkomplexität wuchsen auch die Anforderungen an die Regelwerke der Firewalls und somit auch die notwendige Fachkenntnis der zuständigen Administratoren, um sie zu bedienen. Gerade die kommerziellen Anbieter versuchen ihre Produkte durch das Argument der einfachen Bedienbarkeit und unterstützender Werkzeuge hervorzuheben, doch die nun abstrahierten Konzepte und Mechanismen sind im Detail kaum noch nachvollziehbar und können zu unerwünschten Nebeneffekten führen.

Zusätzlich sind Netzwerkadministratoren und der IT-Support dauerhaft in einer reaktiven, Feuerbekämpfenden Position beim Versuch, die Verfügbarkeit, Zuverlässig-

keit und Leistung ihrer Netzwerke zu sichern, was durch die eingeschränkten Ressourcen Zeit, Geld, Ausrüstung und Personal weiter kompliziert wird.

R. Buchanan schrieb in seinem Buch „The Art of Testing Network Systems“ [Buc96]: „Netzwerküberwachung zeigt an, was passiert ist. Netzwerktests zeigen, was passieren wird.“

Die vorliegende Arbeit beschäftigt sich mit der Modellierung von Paketfiltern als Bestandteil von Firewalls, die eine besondere Art der Netzwerksysteme sind sowie der Entwicklung einer Teststrategie für diese. Damit soll es möglich werden genau überprüfen zu können, was mit einem Paket (oder Datenstrom) beim Passieren der Firewall geschehen wird. Es soll auch helfen, den Vorgang besser zu verstehen, zu dokumentieren und den Administratoren ein fundiertes Werkzeug in die Hand zu geben, um Fehler identifizieren und beseitigen zu können.

1.1. Ziel der Arbeit

Im Rahmen dieser Diplomarbeit wird eine allgemeine Teststrategie für zustandsbehaftete Paketfilter zur praktischen Überprüfung ihrer spezifikationskonformen Funktionsweise in Bezug auf ihre Konfiguration erarbeitet und prototypisch implementiert. Die damit verbundene Fragestellung untersucht den Einfluss der Konfiguration eines Paketfilters auf die von ihm akzeptierten bzw. abgelehnten Protokollflüsse.

Der allgemeine Ansatz soll sicherstellen, dass die Teststrategie nicht nur auf ausgewählte Paketfilter anwendbar, sondern auf typische Paketfilter ausgelegt ist. Die Auswahl der konkret untersuchten Paketfilter soll eine hohe Abdeckung der typischen Paketfilter aufweisen und dabei repräsentative kommerzielle und freie Paketfilter berücksichtigen.

Die Untersuchung beschäftigt sich mit zustandsbehafteten Paketfiltern, die auf den Schichten 3 und 4 des OSI-Referenzmodells [ISO7498-1] wirken. Reine Applikationsprotokollfilter, also Filter auf Schicht 5-7, oder gebündelte Schutzkomponenten sind nicht Bestandteil der Untersuchung, da bereits die Schichten 3 und 4 einen sehr großen Zustandsraum aufspannen, der noch zu untersuchen ist.

Die Aufgaben einer eingesetzten Firewall und des Paketfilters als einer Komponente davon, werden aus dem Schutzbedarf und dem Schutzkonzept, die in den Sicherheitsrichtlinien (security policy) beschrieben sind, abgeleitet. Ein Funktionstest der Firewall müsste dementsprechend die korrekte Umsetzung der Richtlinien testen. Allerdings sind Sicherheitsrichtlinien, wenn sie überhaupt aufgeschrieben werden, oft informal, recht oberflächlich und für das Netzwerk als Einheit definiert, weswegen sie als Basis für eine Teststrategie ungeeignet sind. Die Konfiguration eines Paketfilters muss allerdings immer vorgenommen werden und stellt in vielen Einsatzszenarien die erste vorhandene formale Beschreibung der Aufgabenstellung dar, die einen testbaren Zugriffspunkt beschreibt. Deswegen wird bei dieser Arbeit die Konfiguration als Teil der Spezifikation betrachtet, mit deren Hilfe das zur jeweiligen Protokollspezifikation konforme Verhalten überprüft wird.

Als Grundlage für die Teststrategie soll ein allgemeines Modell für die zustands-

behaftete Paketfilterung und -veränderung erstellt werden. Die Analyse soll auf den Spezifikationen bzw. der offiziellen Dokumentation der Paketfilter basieren, wobei eine Klassifizierung der Verhaltensweisen nach typischen und atypischen Merkmalen durchgeführt werden soll. Letztere benötigen gegebenenfalls eine besondere Berücksichtigung beim Testen, weswegen eine Beschreibungsmöglichkeit entworfen werden soll, mit der das allgemeine Modell und die Teststrategie für einen spezifischen Paketfilter in geeigneter Form vom Anwender eingestellt bzw. angepasst werden kann. Eine möglichst vollständige Abdeckung der identifizierten Merkmale soll erreicht und alle Abweichungen sollen begründet dokumentiert werden.

Schließlich soll eine für die Testumgebung konfigurierbare Teststeuerung entworfen werden, die durch aktives, aber ressourcenschonendes Testen die spezifikationskonforme Funktionsweise des Paketfilters überprüfen kann. Eine Beschreibung der Netzwerkumgebung des Probanden soll berücksichtigt werden, sodass nicht nur Labortests, sondern auch Tests in einer Produktivumgebung möglich werden. Die Aufgabenstellung gibt vor, dass die Teststrategie auf den zwei Werkzeugen FWA und FWTest aufbauen soll, die im Fachbereich Kommunikations- und Betriebssysteme der TU-Berlin entwickelt wurden. Dabei analysiert der FWA die produktabhängige Konfiguration eines Paketfilters und errechnet daraus die zu untersuchenden Eigenschaften. Die Teststrategie soll auf dieser Analyse aufbauen und mit Hilfe von FWTest die Testpakete generieren sowie mit dem Paketfilter kommunizieren.

Die Untersuchungsergebnisse der Teststrategie sollen für den Anwender in unterschiedlichen Detaillierungsgraden aufbereitet werden, um Fehlerquellen, Änderungspotential sowie die Testabdeckung aufzuzeigen.

1.2. Vorgehen und Struktur

Die Aufgabenstellung beinhaltet zwei Aspekte, die zu untersuchen und zu bearbeiten sind – die Untersuchung der Firewallsysteme und die Entwicklung der allgemeinen Teststrategie.

In Abschnitt 2.1 werden allgemeine Netzwerkknoten eingeführt und von den auf Schutzfunktionen spezialisierten Firewallsystemen abgegrenzt. In Abschnitt 2.3 werden mehrere repräsentative Paketfilter ausgewählt, die in den Kapiteln drei bis fünf genauer analysiert werden.

In Kapitel 3 und Kapitel 4 werden die ausgesuchten Firewalls vorgestellt und analysiert. Dabei werden im ersten Teil der Auswertung die unterstützten Merkmale, Dienste und Funktionen der Firewallsysteme betrachtet. Abschnitt 3.3 vergleicht die Mächtigkeit der Probanden und bewertet die Testbarkeit der einzelnen Komponenten sowie ihre Eigenschaften. Der zweite Teil untersucht die Funktionsweise der Paketfilter und konzentriert die Betrachtung in Abschnitt 4.2 auf die zustandsbehaftete Verbindungsverfolgung sowie die Paketflüsse innerhalb der Paketfilter. Dabei wird die grundlegende Problematik der Verbindungsverfolgung zweier Endpunkte in einem Zwischenpunkt, wie es die Firewall ist, erklärt.

In Kapitel 5 wird ein allgemeines Modell für die zustandsbehaftete Paketfilterung

und -veränderung entwickelt. Die zuvor identifizierten Paketflüsse werden in mehrere Arbeitsschritte aufgeteilt und in Abschnitt 5.1 einzeln modelliert. Die Modellierung wird in Abschnitt 5.2 auf besondere Funktionen und in Abschnitt 5.3 auf einen ausgewählten Paketfilter angewendet.

Kapitel 6 führt in den zweiten Aspekt, das Testen, ein. Dort werden allgemeine netzwerk- und firewallspezifische Konzepte und Methoden des Testens vorgestellt.

Eine Teststrategie für die zuvor modellierten Paketfilter wird in Kapitel 7 erarbeitet. Sie wird in Form einer Teststeuerung unter der Bezeichnung FWTStrategy realisiert. Zunächst werden die Anforderungen an die Teststrategie und die Vorbedingungen für die Entwicklung behandelt. Dabei wird das Konzept in den Kontext der Testmethodologie gestellt und neue Begrifflichkeiten eingeführt. Die Teststrategie wird in mehreren Schritten entwickelt. In Abschnitt 7.1 wird mit den Test eines einzelnen Testvektors begonnen, was in Abschnitt 7.2 auf eine vollständige und optimierte Teststrategie übertragen wird.

Auf die Umsetzung der Teststrategie wird in Kapitel 8 eingegangen, das die benötigte Beschreibung des Probanden für die Testkonfiguration und das Modulkonzept des Programms beschreibt. Abschnitt 8.2 dokumentiert die konkrete Umsetzung und die aufgetretenen Probleme der jeweiligen Tests. Einen Ausblick auf künftige Erweiterungen der Teststeuerung gibt Abschnitt 8.3.

Die gesammelten Ergebnisse aus der Entwicklung und Anwendung verschiedener Testreihen mit dem erstellten Prototypen werden in Kapitel 9 beschrieben. Dabei werden zunächst mögliche Anwendungsfälle vorgestellt, um die praktische Relevanz der Teststrategie zu belegen. In Abschnitt 9.2 wird die Untersuchung mehrerer Konfigurationen als Beispiel der Testabdeckung vorgestellt. Erkenntnisse aus der Untersuchung des iptables-Paketfilters werden in Abschnitt 9.3 aufgezeigt.

Kapitel 10 vergleicht die erarbeitete Lösung mit anderen Arbeiten. In Abschnitt 10.1 wird die Implementierung der Teststrategie mit anderen Werkzeugen verglichen. Abschnitt 10.2 dagegen stellt die gesamte Arbeit in den Kontext anderer wissenschaftlicher Untersuchungen. Am Ende werden sowohl die Teststrategie als Konzept als auch FWTStrategy als Werkeug bewertet.

Eine Zusammenfassung der Arbeit und ein Ausblick auf mögliche Fortführung der Arbeit bzw. der Untersuchung werden in Kapitel 11 behandelt.

2. Firewalls

Die vorliegende Arbeit beschäftigt sich mit der Untersuchung von zustandsbehafteten Paketfiltern. Dafür ist es notwendig diese Klasse von Netzwerkgeräten genau zu erfassen und von anderen Geräten abzugrenzen. In diesem Kapitel werden zwei Aspekte vorgestellt, die zur Klassifizierung der Netzwerkgeräte und deren Merkmale verwendet werden.

In Abschnitt 2.1 werden verschiedene Netzwerkgeräte und deren Funktionen eingeführt. Darunter werden verschiedene Firewalltypen, also Netzwerkknoten mit Schutzaufgaben, betrachtet.

Abschnitt 2.2 stellt allgemeine Sicherheitsziele der Informationssicherheit sowie die speziellen Bedrohungen gegen Netzwerkknoten vor. Die Sicherheitsziele und mögliche Bedrohungen werden zur Bewertung und Einordnung der Firewalldienste und -merkmale verwendet.

Im letzten Abschnitt (2.3) werden typische bzw. repräsentative Paketfilter ausgewählt. Sofern der Paketfilter nur ein Bestandteil des Systems ist, wird das Produkt den vorgestellten Firewalltypen zugeordnet.

2.1. Netzwerkfunktionen und Firewalltypen

Netzwerke, wie sie hier betrachtet werden, bestehen aus adressierbaren Netzwerkgeräten, die untereinander Daten austauschen können. Die Kopplung der verschiedenen Teilnehmer und Bereiche kann gemäß dem Open Systems Interconnection (OSI) Referenzmodell [ISO7498-1], das ein geschichtetes und abstraktes Modell zur herstellerunabhängigen Beschreibung von Kommunikations- und Computernetzwerkprotokollen beschreibt, auf den Schichten 1 bis 4 geschehen. Dementsprechend existieren auch schichtspezifisch unterschiedliche Geräte, die diese Aufgabe erfüllen. In Abbildung 2.1 wird das OSI-Referenzmodell dem TCP/IP-Modell [RFC1122] gegenüber gestellt. Obwohl die hier behandelten Protokolle treffender dem TCP/IP-Modell zuzuordnen wären, wird für die Diskussion das allgemeinere OSI-Referenzmodell verwendet.

Für die folgenden Betrachtungen und die Einführung in Firewalls sind die Schichten 3 (Vermittlung) und 4 (Transport) des OSI Referenzmodells interessant. Geräte der unteren Schichten, zu denen z.B. Hubs, Repeater, Bridges, Switches und WLAN Access Points gehören, werden nur zur Abgrenzung genannt.

Router arbeiten auf Schicht 3 und besitzen für jedes angeschlossene Netz eine (zumindest logische) Schnittstelle, um zwischen den Netzen zu vermitteln. Die Weiterleitungsregeln für die Adressaten werden einer Routingtabelle entnommen. Router arbeiten medienunabhängig und können an den Schnittstellen unterschiedliche Netz-

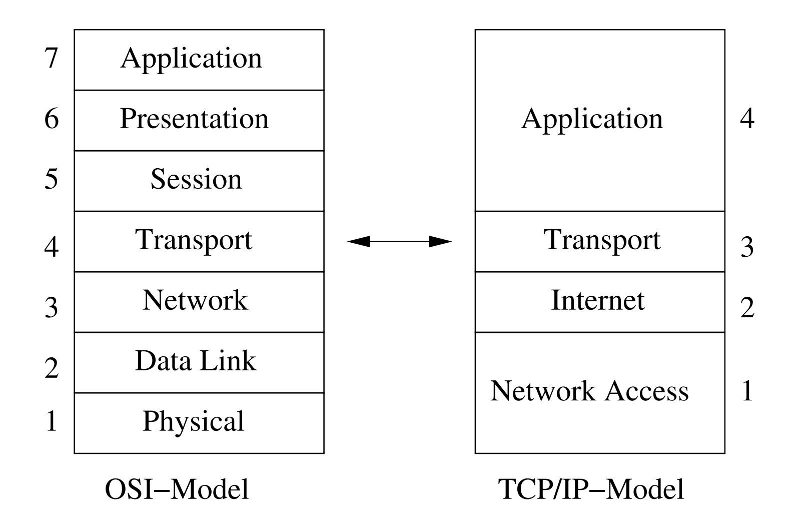


Abbildung 2.1: Das OSI und das TCP/IP Referenzmodell im Vergleich

techniken verwenden. Dafür müssen aber die weiterzuleitenden Protokolle der Schicht 3 bekannt sein, weswegen Router protokollabhängig arbeiten. Ein Router, der mehrere Protokolle weiterleiten kann (z. B. IP und IPX), wird auch Multi Protocol Router genannt.

Gateways werden im allgemeinen Sprachgebrauch oftmals mit Routern gleichgesetzt, obwohl ein Gateway im Gegensatz zum Router auf allen Schichten implementiert werden kann und in der Lage ist, auch zwischen verschiedenen Protokollen zu vermitteln. Dementsprechend lautet die deutsche Bezeichnung auch „Protokollumsetzer“. In einigen Betriebssystemen wird die IP-Adresse des default routers auch als Gateway bezeichnet. In dieser Arbeit wird Gateway im Sinne der ersten Definition benutzt.

Bei der Besprechung der Protokolle und der Netzwerkkomponenten ist es notwendig eine Relation zwischen den beiden Endpunkten zu betrachten. Der intuitive Begriff *Verbindung* wird sehr stark mit verbindungsorientierten Protokollen wie TCP verknüpft und wäre bei paketorientierten Protokollen wie UDP oder ICMP missverständlich. Deswegen wird hier protokollübergreifend von *Assoziation* gesprochen.

Netzwerkknoten mit Schutzaufgaben

Eine *Firewall* ist ein Netzwerkvermittlungsknoten, der am Rand von zwei oder mehreren Netzwerken den durchgehenden Netzwerkverkehr reglementiert und Sicherheitsaufgaben übernimmt. Zu den grundlegenden Aufgaben einer Firewall gehört der Schutz der Netzwerkknoten, die durch die Firewall verbunden sind und entsprechend der Sicherheitspolitik schützenwerten sind. Die meisten Firewalls übernehmen auch Routing- und Protokollumsetzungsaufgaben. Eine Firewall ist aber nicht nur passiver Beobachter mit Weiterleitungsfunktion, sondern muss aufgrund seiner Aufgabe auch aktiv eingreifen. Dabei dienen Protokollspezifikationen und die Konfiguration

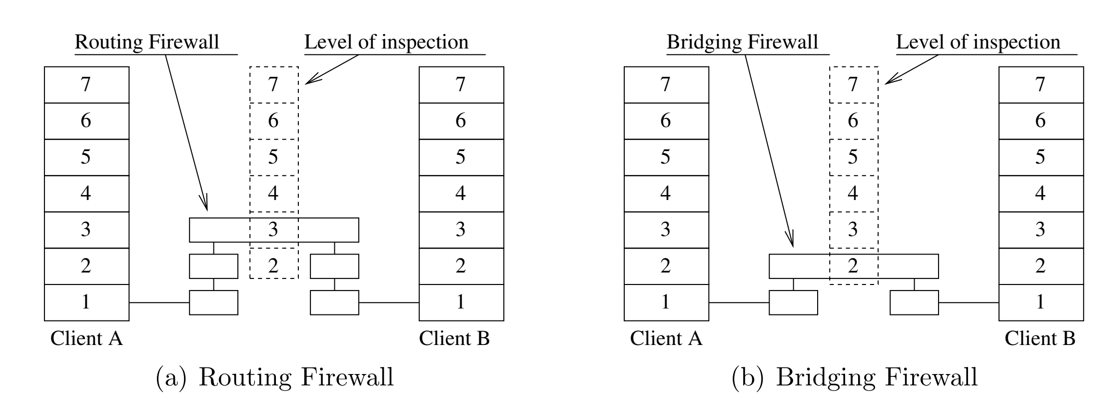
als Grundlage für die Eingriffsentscheidungen.

Genauer betrachtet ist eine Firewall ein *Firewallsystem*, das aus mehreren Komponenten besteht. Deswegen müssen die Begriffe Firewallsystem, Firewall-Software und Filtersoftware eingeführt und unterschieden werden. Firewallsystem bezeichnet das gesamte System bestehend aus allen Hard- und Software-Komponenten. Kern eines Firewallsystems ist ein Paketfilter, der für die Reglementierung des Netzwerkverkehrs zuständig ist. Der Paketfilter wird für die Filterung von unerwünschtem Datenverkehr zwischen Bereichen mit unterschiedlichen Sicherheitsanforderungen genutzt. Neben ihm können noch weitere Software-Komponenten installiert sein, die zusätzliche Dienste und Funktionen anbieten.

Eine Firewall kann ihre Schutzfunktion auf einem Endsystem oder einem vermittelndem System verrichten. Dabei sind die Varianten, die auf einem Endsystem zum Einsatz kommen, zusätzlich zu unterteilen. Die softwarebasierenden *Personal Firewalls* haben keine vermittelnden Funktionen und reglementieren vielmehr den eingehenden sowie den von den lokalen Anwendungen ausgehenden Verkehr. Zusätzlich werden sie auch zum Schutz vor Viren, Würmern und Trojanern eingesetzt[1]. Auf Endsystemen kann aber auch die Firewallsoftware der vermittelnden Systeme laufen, die die Funktionen der vermittelnden Firewalls bietet. Im Folgenden werden nur noch vermittelnde Firewalls betrachtet.

Geschichtlich betrachtet basierten die ersten Firewalls auf intelligenten Routern, die ausgewählte Pakete weiterleiteten[2]. Die Router wurden immer weiter um ausgeklügeltere Filtermechanismen erweitert. Später wurden spezialisierte Firewallsysteme entwickelt, die notwendigerweise auch Routingaufgaben übernahmen. Deswegen existiert keine eindeutige Grenze zwischen Firewalls und Routern.

Eine Firewall, im Sinne einer allgemeinen Schutzkomponente, kann auf jeder Ebene des OSI-Schichtenmodells implementiert werden. Dabei trennt eine Firewall der Schicht n die Schichten $1 - n_{-1}$ und verbindet die n-te Schicht.



(a) Routing Firewall

(b) Bridging Firewall

Abbildung 2.2: Firewall-Betriebsmodi im Vergleich

[1] Die Personal Firewall wird auf den Systemen vieler Heimbenutzer eingesetzt. Sie wird aufgrund der abweichenden Zielfunktion in dieser Arbeit nicht betrachtet.

[2] Zur Firewallgeschichte vergleiche [IF02] und [Spe06, Seiten 37–44].

In Abbildung 2.2 sind die meist eingesetzten Routing- und Bridgemodi abgebildet. Im ersten Fall verhält sich die Firewall wie ein Router, wobei jeder Netzwerkadapter eine IP-Adresse bekommt und darüber ansprechbar ist. Im Bridgemodus funktioniert die Firewall wie ein Switch bzw. eine Bridge. Letzteres hat den Vorteil, dass die Firewall nicht direkt ansprechbar und dadurch auch schwerer angreifbar ist. Die Inspektionstiefe bei der Filterung ist unabhängig von der Betriebsart der Firewall.

Die Art und Tiefe der Inspektion hat sich mit der Zeit weiter entwickelt, so dass nun verschiedene Schutzkonzepte nebeneinander existieren und sich ergänzen. Tabelle 2.1 fasst die Arten zusammen und schlüsselt die untersuchten OSI-Ebenen mit ihren Eigenschaften auf (vgl. hierzu auch [ZCC00, Kap. 5] und [Spe06, Kap. 2]).

	OSI-Layer			
Typ	3	4	5-7	untersucht ...
stateless filter	✓	✓	–	IP, Ports
stateful filter	✓	✓	–	IP, Ports, Verbindungszustand
Proxy, ALG	–	–	✓	Anwendungsspezifische Untersuchungen
IDS/IPS	–	–	✓	Signaturen, Heuristiken, Anomalieerkennung
Deep Packet Inspection	✓	✓	✓	Stateful Filtering + IDS/IPS
UTM	✓	✓	✓	DPI + Anti-Virus

Tabelle 2.1: Unterscheidungsmerkmale von Firewall-Inspektionstypen

Die frühen Paketfilter beherrschen das sogenannte *stateless filtering*, untersuchen die OSI-Schichten 3 und 4 und werten statische Eigenschaften jedes einzelnen Paketes kontextfrei aus. Einige Varianten unterstützen auch die Untersuchung der Schicht 2.

Später wurden zustandsbehaftete Paketfilter (*stateful filtering*) entwickelt, die im Vergleich zum stateless filter auch dynamische Eigenschaften der OSI-Schicht 4 berücksichtigen. Im Gegensatz zu den einfachen Paketfiltern haben sie eine (begrenzte) Vorstellung von einer Assoziation und dem Protokollfluss zwischen den Endpunkten, wodurch sie abhängig von der Phase der Assoziation gezielter reglementieren können. Neuerdings wurden auch die Bezeichnungen *stateful inspection* bzw. *deep inspection* für Firewalls eingeführt, die neben dem stateful filtering auch die anwendungsspezifischen Datenbereiche der Pakete auf Konformität bzw. atypische Muster untersuchen [Ran05].

Ergänzend zu den Paketfiltern wurden Proxy-basierende Applikationsfilter (ALG, *Application Level Gateway*) eingesetzt, die den Netzwerkverkehr der OSI-Schichten 5 bis 7 auswerten, also den Protokollfluss auf der Anwendungsebene analysieren und gegebenenfalls bereinigt weiterleiten oder unterbrechen können. ALGs terminieren den ankommenden Datenverkehr und bauen eine zweite Verbindung zum eigentlichen Zielsystem auf. Im Gegensatz zu einem Paketfilter muss ein ALG das vermittelte Protokoll vollständig verstehen, um die neue Verbindung aufbauen zu können. ALGs können auch zusätzliche Funktionen wie Caching, Logging, suchwortbasierende Inhaltskontrolle (z.B. Java und ActiveX entfernen, gewalt- oder sexuellorientierte Inhalte blockieren), Anti-Virus-Maßnahmen sowie Authentifizierung anbieten. Manche SSL-Proxies werden als vorgeschalteter Terminierungspunkt für interne Server eingesetzt, wodurch sie sich für die Server ausgeben und auch verschlüsselten Da-

tenverkehr untersuchen können. Dadurch wird aber die Ende-zu-Ende-Semantik von verschlüsselten Kanälen aufgehoben.

Eine andere Ebene der Inspektion bieten *Intrusion Detection* (IDS) bzw. *Intrusion Prevention Systeme* (IPS). Während die ersteren einen erfolgten Angriff feststellen und darüber informieren können, sind letztere in der Lage, gerade auftretende Angriffsmuster zu erkennen und sie mit Hilfe des Paketfilters zu unterbrechen. ID&P-Systeme setzen zwei Klassen von Techniken ein[3]. Zur Angriffserkennung vergleichen sie den Datenverkehr mit bekannten Signaturen. Bei der Anomalieerkennung untersuchen sie die Daten auf Abweichungen zu den gesammelten netzwerkspezifischen Daten. ID&P-Systeme werden in netzwerk- und hostbasierte Systeme unterteilt. Netzwerkbasierte Systeme schützen die verbundenen Systeme und letztere das eigene Wirtssystem.

Die neueste Generation von Sicherheitsprodukten vereinigt alle bisherigen Funktionen zusammen mit einer Anti-Viren-Komponente in einem Produkt und wird unter der Bezeichung *Unified Threat Management* (UTM) beworben.

Network Address and Port Translation

Bei der Recherche zu dieser Arbeit hat sich Network Address Translation (NAT) als eines der unübersichtlichsten Netzwerkkonzepte herausgestellt. Obwohl die Kernfunktionalität überall gleich vorgestellt wurde, weichen die gewählten Bezeichnungen der einzelnen Varianten stark voneinander ab. Um eine Vergleichbarkeit zu gewährleisten, werden die verschiedenen Varianten und Bezeichnungen aus [RFC2663; RFC3022] zusammengefasst vorgestellt. [RFC3489] führt eine abstraktere Sicht auf die NAT-Arten ein, die in dieser Arbeit aber nicht verwendet wird.

Network Address Translation (NAT) wird häufig in die Gruppe der Firewalltechnologien eingereiht[4], wurde jedoch zunächst als Antwort auf die beschränkte Anzahl von IPv4-Adressen im Internet entwickelt. Es ermöglicht die Abbildung einer Menge von Netzwerkadressen auf eine andere. Wird dabei die Quelladresse verändert, so wird von *source NAT* (SNAT) gesprochen. Die Änderung der Zieladresse wird als *destination NAT* (DNAT) bezeichnet. Gleiche Abbildungen sind auch mit den Portadressen möglich, was als Port Address Translation (PAT) bezeichnet wird. NAPT beschreibt die gleichzeitige Änderung von Adresse und Port. Diese ist besonders mächtig in Verbindung mit Policy NAT, bei der die Abbildung nur unter Berücksichtigung der Quell- und Zieladresse bzw. der Ports durchgeführt wird.

Bis auf wenige untypische Spezialfälle benötigen alle NAT-Varianten eine zustandsbehaftete Verbindungsverfolgung, um eine bidirektionale Kommunikation unterstützen und die aktuellen Zuordnungen auflösen zu können. Eine allgemeine NAT-Umsetzung ist deshalb ohne Zustandsaufzeichnung kaum sinnvoll.

Bei der Abbildung einer Adress- und Portmenge auf eine andere können unterschiedliche Mechanismen und Strategien benutzt werden. Im einfachsten Fall sind

[3]Eine detailierte Einführung in die Funktionsweise und den Betrieb von NIDS und IPS kann von [BSI02] angefordert oder online abgerufen werden.
[4]Hierzu vergleiche z.B. [Spe06, Kapitel 2].

beide Mengen gleich groß und jedes Element aus Menge A wird statisch und linear auf ein Element der Menge B abgebildet. Sind die Mengen ungleich geschaffen, können verschiedene dynamische Abbildungen eingesetzt werden, die die gerade unbenutzten Elemente berücksichtigen. Dabei gibt es u.a. Strategien, die versuchen:

- bei SNAT für ein System immer die gleiche Quelladresse beizubehalten oder
- die Abbildung möglichst zufällig zu wählen,
- bei DNAT zu einer Gruppe die Auslastung gleich zu verteilen (load balancing).

Eine besondere Variante der dynamischen source NAPT ist auch als *masquarading* oder *hide NAT* bekannt, die dazu verhilft den internen Netzaufbau zu verschleiern oder einen privaten Adressraum auf einen öffentlichen Adressraum abzubilden. Dabei werden die Quelladresse und -port aller durch die Firewall oder den Router ausgehenden Pakete durch eine der externen IP-Adressen und einen ungenutzten Port des Routers bzw. der Firewall selbst ersetzt. Die Zuordenbarkeit der Verbindungspartner bleibt aufgrund des eindeutigen, ungenutzten, lokalen Ports weiterhin erhalten.

Weitere Szenarien, für die NAT verwendet werden kann, sind die Auflösung von Adresskonflikten und Überlappungen von Adressbereichen, die Realisierung transparenter Proxies oder die (temporäre) Umleitung von Anfragen auf ein anderes System.

Besonders zu beachten ist, dass sämtliche Formen von NAT nicht nur für die Adressen im IP-Header durchgeführt werden müssen, woraus sich verschiedene Komplikationen ergeben (vgl. [RFC3027]). Anzupassen sind auch gegebenenfalls Routingangaben in den IP-Optionen und ICMP-Fehlernachrichten. Auch viele Anwendungsprotokolle, die zusätzliche Verbindungskanäle vereinbaren, benutzen die lokal bekannten Adressen und müssen ebenfalls angepasst werden – vor allem, wenn sie selbst keine Mechanismen bereitstellen, mit NAT umzugehen. Als Beispiele wären hier Protokolle aus den Bereichen VoIP (SIP, h323), Dateitausch (FTP, IRC DCC) und zahlreiche Peer-to-Peer Anwendungen zu nennen. Diese Funktionalität kann in Form von Erweiterungen für den Paketfilter bereit gestellt oder in einer Proxy-Anwendung realisiert werden.

2.2. Sicherheitsziele und deren Bedrohungen

Zu den Grundprinzipien und Zielen der Informationssicherheit gehört die Sicherung der Vertraulichkeit, der Integrität und der Verfügbarkeit[5]. Zusätzlich existieren mehrere Erweiterungen des Modells. Tabelle 2.2 zeigt eine Erweiterung der Aspekte um Zurechenbarkeit und Zugriffskontrolle nach [Sch03, Kapitel 1][6]. Zugriffskontrolle wird mit Hilfe der Authorisierung umgesetzt, die die Identifikation und die Authentifikation des zugreifenden Subjektes voraussetzt. Jede so gestattete oder verbotene Aktion

[5] Im englischen Sprachraum als CIA-Triade – Confidentiality, Integrity, Availability – bekannt.
[6] Tabelle ist zusätzlich mit den Angriffsbeispielen erweitert.

muss einem Akteur zurechenbar und im strengeren Sinne sollte die Aktionsbelegung auch nicht anfechtbar sein.

Die Sicherheitsziele werden durch die Enthüllung bzw. Preisgabe von Informationen, die Korruption bzw. Veränderung von Daten, die Unterbrechung bzw. Störung, die Täuschung und die widerrechtliche Aneignung bedroht. Die Sicherheitsziele und dessen Bedrohungen werden in dieser Arbeit als Grundlage verwendet, verschiedene Angriffsklassen vorzustellen und die Sicherheitsmechanismen der Firewalls vorzustellen.

Technische Sicherheits–ziele	**Technische Bedrohungen**						
	Verschleierung Maskarade	Abhören	Verletzung d. Autorisierung	Verlust, Modifi–kation von Daten	Fälschung von Daten	Abstreiten v. Kommunikation	Sabotage, Störung
Vertraulichkeit	✓	✓	✓				
Integrität	✓		✓	✓	✓		
Verfügbarkeit	✓		✓	✓			✓
Zurechenbarkeit	✓		✓		✓	✓	
Zugriffskontrolle	✓		✓		✓		
	Angriffsbeispiele auf Netzwerk– und Transportebene						
	Adressfälschung	wire–tapping, sniffing	Erkundung von Systemen/ Netzwerken	Session–Hijacking Packet injection			PoD, Winnuke, Teardrop, Flooding, DoS

Tabelle 2.2: Sicherheitsziele und Angriffsklassen

IP-Spoofing oder Adressfälschung bedroht unmittelbar die Zurechenbarkeit von Aktionen. Indirekt hat die Verschleierung bzw. Maskarade auch Auswirkungen auf die anderen Sicherheitsziele. Auch das Abstreiten von Kommunikationsakten gefährdet die Zurechenbarkeit.

Verschiedene Techniken zum Abhören von Daten gefährden die Vertraulichkeit von Informationen. Als vertraulich und sicherheitsrelevant können z.B. Informationen über die Netzwerkinfrastruktur eingestuft werden. Deswegen ist auch die Netzwerk- und Systemerkundung, zu der das Sammeln von IP-Adressen, das Scannen nach offenen Ports oder die Feststellung der eingesetzten Software gehören, dieser Kategorie zuzuordnen. Der Akt des Belauschens kann auch eine Verletzung der Autorisierung bedeuten.

Der Verlust, die Modifikation oder die Fälschung von Daten gefährden die Integrität. Gleichzeitig sind darüber auch widerrechtliche Zugriffe möglich, die nicht mehr korrekt zuordenbar sind. Dadurch kann auch die Verfügbarkeit gestört werden.

Das am meisten bedrohte Sicherheitsziel in Netzwerken ist die Verfügbarkeit von Systemen und Diensten. Sie kann durch präparierte Pakete, die die Systeme zum Stillstand oder Absturz bringen, oder die Überlastung der verfügbaren Ressourcen angegriffen werden. Die Angriffe gegen die Verfügbarkeit werden als Denial of Service (DoS) bezeichnet.

Gezielt eingefügte Fehler in den Protokollen IP, TCP, UDP oder ICMP lassen sich für betriebssystemspezifische DoS-Angriffe nutzen[7]. Dazu zählen u.a. Ping of Death (PoD), die Teardrop Attacke oder WinNuke.

[7] Vgl. hierzu `http://de.wikipedia.org/wiki/Denial_of_Service`.

Die überdurchschnittliche Beanspruchung der Ressourcen wird i.d.R. erreicht, wenn mehr Anfragen generiert werden als der Dienst bearbeiten kann, weswegen er sie verwerfen muss. Eine Möglichkeit für so einen Angriff ist das SYN-Flooding, das die erste Phase des 3-Wege-Handshakes bei TCP von nicht antwortenden Quelladressen vortäuscht und so die SYN-Backlog-Tabellen des Zielsystems, in denen unvollständige TCP-Verbindungsversuche verwaltet werden, überflutet. Ähnliche Flooding-Angriffe sind auch mit ICMP und UDP möglich.

Die Verfügbarkeit der Dienste kann gestört werden, indem das Zielsystem oder die vermittelnden Knoten angegriffen werden. Deswegen zielen auch manche Angriffe auf die Firewalls selbst, die ebenfalls eine Angriffsfläche bieten, auch wenn sie darauf abgestimmt sind Angriffen und Überlastungen entgegenzuwirken. Mit dem Ausfall der Firewall würden auch die Erreichbarkeit und der Dienst aller über sie verbundenen Systeme wegfallen. Deswegen ist ein Teil der Firewallfunktionen auch dafür gedacht, den Betrieb zu sichern und die Verfügbarkeit zu erhöhen.

Die Merkmale der Firewalls können als Methoden und Strategien angesehen werden die Sicherheitsziele und die Sicherheitsrichtlinien durchzusetzen. Die hier untersuchten Paketfilter dienen der Umsetzung der Zugriffskontrolle auf Systeme und Dienste. Indirekt können sie auch die Zurechenbarkeit (z.B. über das Logging der Zugriffe), die Verfügbarkeit (z.B. über die Reglementierung der übermäßigen Dienstnutzung oder das Filtern von Angriffen auf den IP-Stack), die Vertraulichkeit der Netzwerkstruktur (über NAPT) und beschränkt auch die Integrität der Daten (z.B. durch die Überprüfung der Checksummen und der Sequenznummern) umsetzen.

Angriffe auf die Vermittlungs- und Transportschicht gehören zu dem Standardrepertoire der verfügbaren Angriffswerkzeuge und können inzwischen von allen Paketfiltern abgewehrt wrden. Zu diesen Angriffen gehört Adressfälschung, Erkundung von Netzwerken, Session-Hijacking bzw. Packet injection und verschiedene Sabotage-Akte. Weitere, und immer noch erfolgreiche, Angriffe nutzen Schwächen der Dienste und der Applikationen aus. Deswegen ist für eine striktere Umsetzung der Sicherheitsziele und Abwehr der Angriffe eine Kombination mit den anderen vorgestellten Schutzkomponenten und -konzepten notwendig, was den Trend zur Integration der Komponenten in ein Unified Threat Management unterstützt.

Sicherheitsrichtlinien

Der Schutzbedarf und die Schutzmaßnahmen für eine organisatorische Einheit werden in sogenannten *Sicherheitsrichtlinien* (engl. security policy) definiert. Nach [BSI06] werden Sicherheitsrichtlinien mehrstufig aufgebaut. Die oberste Leitlinie wird i.d.R. von der Geschäftsleitung als Teil der unternehmensweiten allgemeinen Sicherheitsziele, die auch Datenschutz, Personensicherheit, Geräteschutz, Computer- und Netzwerksicherheit beinhaltet, aufgestellt. Diese Richtlinien sind meist so informell sowie allgemein genug und unabhängig von einer konkreten Umsetzung gefasst, dass Änderungen selten notwendig werden und eine technische Umsetzung nicht vorgegeben wird. Trotz der Allgemeinheit sind Sanktionen ein essentieller Bestandteil solcher Richtlinien, wobei auch eine Balance zwischen dem finanziellem Aufwand und der

Sicherheitssteigerung erreicht werden soll. Detaillierter als die Leitlinie wird die allgemeine Sicherheitskonzeption gefasst. Sie beschreibt die ausführlichen Anforderungen und die dazugehörenden Maßnahmen. Durch den höheren Detaillierungsgrad sind Änderungen häufiger möglich. In der dritten Ebene werden technische Details, konkrete Maßnahmen und produktspezifische Einstellungen beschrieben. Sie enthält viele Dokumente, die regelmäßig geändert und typischerweise nur von den zuständigen Fachpersonen gelesen werden. Darunter fällt auch ein Firewall-Regelwerk, das sehr konkret formuliert und an eine Netzwerkstruktur bzw. ein Gerät gebunden ist.



Abbildung 2.3: Hierarchischer Aufbau von Richtlinien nach IT-Grundschutzkatalogen, Maßnahme M2.338

2.3. Auswahl und Klassifizierung repräsentativer Firewallsysteme

Bei der Untersuchung werden sechs Firewallsysteme betrachtet, von denen die ersten vier Vertreter – netfilter/iptables, IP FireWall, IPFilter und Packet Filter – zustandsbehaftete Paketfilter sind. Sie sind auch alle im Quellcode verfügbar, was die detaillierte Untersuchung möglich macht. Zusätzlich werden zwei kommerzielle Produkte untersucht, die der Kategorie deep inspection filter zuzuordnen sind.

Die Firewalls wurden aufgrund ihrer Verfügbarkeit zum Untersuchungszeitpunkt ausgewählt. Es wird aber auch angenommen, dass sie repräsentativ für typisch eingesetzte Systeme sind. Genaue Informationen zur Marktverteilung von Paketfiltern konnten nicht ermittelt werden. Allerdings werden bei der Untersuchung alle aktuellen Open Source Paketfilter betrachtet. Laut einer Schätzung vom OSS-Experten Harald Welte [Wel07] wird bei über der Hälfte der seit 2003 produzierten eingebetteten Netzwerkgeräte Linux verwendet – und demnach auch netfilter/iptables, sofern ein Paketfilter Teil des Funktionsumfangs ist. Sowohl Linux wie auch die BSD-Systeme sind als Basis für Internet-Server sehr beliebt.

Anbieter	Produktname	Version	seit	Plattform/OS	Lizenz
netfilter.org	netfilter	Linux 2.6.20	1999	Linux	GPL
freebsd.org	IP FireWall	FreeBSD 7	1993	FreeBSD, MacOS X, (BSD/OS)	BSD
openbsd.org	Packet Filter	OpenBSD 4.1	2001	OpenBSD, FreeBSD, NetBSD, DragonFly	BSD
Darren Reed	IPFilter	4.1.19	1993	verschiedene UNIX-Varianten	IPFilter
Cisco Systems	PIX	6.3	1994	HW + OS (Finesse)	kommerziell
CheckPoint Software	Firewall-1	SPLAT R60	1994	HW Bundle, SecurePlatform	kommerziell

Tabelle 2.3: Vergleich der betrachteten Firewalls

Die beiden kommerziellen Vertreter gehörten laut IDC im Jahr 2004 zu den fünf stärksten Anbietern im westeuropäischen Gebiet[8].

Netfilter [Netfilter] ist eine Kernel-Schnittstelle zur Reglementierung von Paketflüssen und bildet die Basis für alle Firewallfunktionen unter Linux. Netfilter existiert seit 1999 und nutzt die mit dem bis zur Kernelserie 2.3 eingesetzten Firewallsystem ipchains gewonnenen Erfahrungen, um dessen Schwächen zu beseitigen. Die Regeln werden in Tabellen abgelegt und je nach Betriebsmodus der Firewall über die Kommandozeilenprogramme iptables bzw. ip6tables sowie ebtables und arptables konfiguriert. Die beiden ersteren sind Bestandteil des iptables-Paketes und dienen zur Reglementierung von IPv4 und IPv6 im Routingmodus. Ebtables und arptables sind im ebtables-Paket enthalten und ermöglichen die Konfiguration einer bridging Firewall.

IP FireWall [IPFW] bezeichnet sowohl den Filter-Code im Kernel als auch das Kommandozeilenprogramm IPFW zum Einstellen der Filterregeln. IPFW gehört zu den ältesten Unix-Firewalls, weswegen die ersten Quellen auch auf das Jahr 1993 und BSD/OS der Firma BSDi zurückzuführen sind, wo es bis zum Ende der Distribution im Jahre 2003 weiterentwickelt wurde. IPFW wurde im November 1994 offiziell in FreeBSD 2.0 eingeführt und ist heute einer von drei Paketfiltern des Betriebssystems. Im Sommer 2002 ist nach einer grundlegenden Überarbeitung IPFW2, welches um IPv6 und zustandsbehaftete Filterung erweitert wurde, erschienen. Weiterhin basiert die in Mac OS X eingesetzte Filtersoftware ebenfalls auf IPFW2. In dieser Arbeit wird aufgrund der fehlenden Dokumentation der BSDi-Version vorrangig die FreeBSD-Variante vorgestellt. Dort, wo es möglich ist, werden die Unterschiede der beiden gegenüber gestellt.

IPFilter [IPF] wurde im Jahre 1993 als alleinstehendes Projekt gestartet und zuerst 1996 in NetBSD 1.2 bzw. FreeBSD 2.2 integriert, lässt sich aber auch in zahlreiche

[8] Die Angaben sind über `http://www.channelpartner.de/news/207172/index.html` und `http://www.channelpartner.de/news/205521/index.html` referenziert, da der direkte Bezug von IDC kostenpflichtig ist. Aktuellere Zahlen konnten nicht ermittelt werden.

weitere UNIX-verwandte Systeme integrieren. Es stellt u.a. die Basis für die Firewallsoftware in Sun Solaris 10 bzw. OpenSolaris oder die kommerzielle Sidewinder G2 Security Appliance[9] von Secure Computing dar.

Der Packet Filter [PF] wurde 2001 für OpenBSD 3.0 entwickelt, nachdem sich lizenztechnische Probleme mit dem bis dahin eingesetzen IPF ergeben hatten. Später wurde es auch nach FreeBSD und NetBSD portiert, wo es als alternativer Paketfilter zur Verfügung steht. Es ist in Syntax und Funktionsumfang mit IPF vergleichbar.

Die Private Internet eXchange Security Appliance, kurz PIX, ist das meistverkaufte[10] Firewallsystem bei Cisco Systems. Es ist eine Kombination aus sicherheitsorientierter Hardware und der echtzeitfähigen PIX Systemsoftware, die in verschiedenen Ausbaustufen angeboten werden. Die Untersuchung basiert auf der Dokumentation der Systemsoftware Version 6.3 und kann für neuere Versionen abweichen, da neuere Softwareversionen der PIX 7.x-Reihe auf der Architektur der Cisco Adaptive Security Appliance (ASA 55xx) Firewallsysteme basieren.

CheckPoint FireWall-1 war die erste kommerzielle Firewall mit einer vereinfachten Konfiguration über eine grafische Oberfläche, weswegen sie seit ihrer Einführung 1994 sehr schnell große Verbreitung fand. Zudem basiert das Konzept der zustandsbehafteten Untersuchung von Paketen auf der von CheckPoint 1997 patentierten *stateful inspection* und deren INSPECT-Engine. Die folgenden Beschreibungen beziehen sich auf die Dokumentation der FireWall-1 NGX auf der Secure Platform R60[11].

[9] Die Merkmale und Funktionen der Sidewinder Firewall wurden ebenfalls ausgewertet, werden jedoch nicht separat dargestellt. Für eine detailierte Darstellung vgl. `http://www.sidewinder.com`.

[10] Vgl. `http://www.cisco.com/en/US/products/hw/vpndevc/ps2030/index.html` und `http://www.networkworld.com/community/?q=node/12346`.

[11] Siehe `http://www.checkpoint.com/products/secureplatform/index.html`

3. Merkmale der Firewallsysteme

In diesem Kapitel werden die in Abschnitt 2.3 ausgewählten Firewallsysteme weiter analysiert. Aus allen Merkmalen werden hier vor allem die Sicherheits- und die Netzwerkfunktionen (Abschnitt 3.1) betrachtet. Weitere zusätzliche Sicherheitsdienste wie Authentifikation, ID&P-Systeme, Proxy-Funktionen, Content-Filter oder Anti-Viren-Programme werden aufgrund des Betriebes auf der Anwendungsschicht nicht weiter klassifiziert. Bei den kommerziellen Anbietern kann aber die Art und Anzahl solcher integrierten Dienste und der Managementfunktionen als weiteres Unterscheidungsmerkmal herangezogen werden. Die vorgestellten freien Produkte sind nicht weniger mächtig, die einzelnen Komponenten müssen aber seperat ausgesucht und in Betrieb genommen werden.

Im Hinblick auf die Tests der Paketfilter werden in Abschnitt 3.2 die Merkmale der Regelverarbeitungs- und Filtereinheiten vorgestellt. Die Testbarkeit der vorgestellten Merkmale wird in Abschnitt 3.3 erläutert und bewertet.

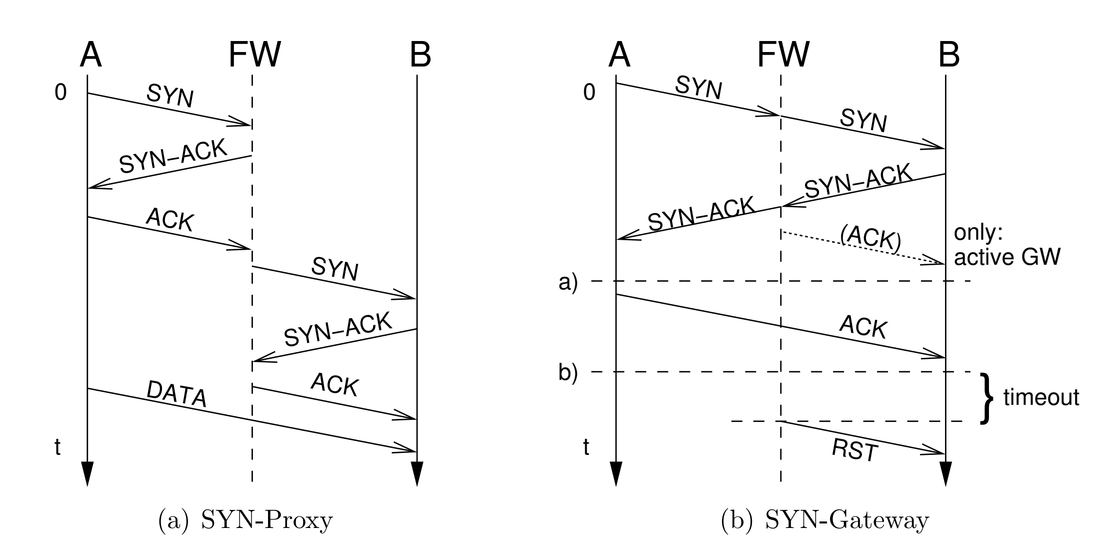
3.1. Sicherheits- und Netzwerkfunktionen

Die Klassifizierung und Vorstellung der Sicherheitsfunktionen der Firewallsysteme orientiert sich an den in Abschnitt 2.2 vorgestellten Sicherheitszielen. Die meisten Merkmale unterstützen die Sicherung der Verfügbarkeit. Dies wird vor allem durch verschiedene Mechanismen der Zugriffskontrolle erreicht. Vertraulichkeit, Integrität und Zurechenbarkeit werden vor allem durch den Einsatz von kryptopgrafischen Protokollen umgesetzt.

Verfügbarkeit der Dienste

Firewalls setzen Mechanismen ein, mit denen sie die Verfügbarkeit der anderen Knoten erhöhen können. Dazu können sie z.B. stellvertretend für die Dienste antworten und den Verbindungsversuch verifizieren (SYN-Proxy), mit einem verkürztem Timeout die Überlastung verhindern (SYN-Gateway), die Quelladressen auf Erreichbarkeit prüfen (Anti-Spoofing), Beschränkungen bezüglich der Verbindungsversuche pro Zeiteinheit setzen und strikte Timeouts einhalten.

Der Einsatz eines SYN-Proxies für TCP-Verbindungen verhindert das Überlaufen der Zustandstabelle durch Verbindungsversuche von gefälschten Adressen. Nach einem erfolgreichen und legitimen Verbindungsaufbau werden weitere Pakete nur noch mit der Zustandstabelle und den dort gespeicherten Daten verglichen. Von den untersuchten Firewalls bieten nur Checkpoint FireWall-1 und Packet Filter einen SYN-Proxy – erste sogar die zwei Mechanismen in Abbildung 3.1.



(a) SYN-Proxy

(b) SYN-Gateway

Abbildung 3.1: Schutzvarianten gegen SYN-Flooding

Damit die Firewall nicht auch überlastet wird, neue Verbindungsversuche ablehnt und die Diensterreichbarkeit beeinträchtigt, verfügt sie über Mechanismen, die im Falle einer besonderen Auslastung durch DoS-Angriffe Maßnahmen einleiten. [Gil02] und [Sch97] beschreiben solche Angriffe, zu denen u.a. das Flooding der Verbindungstabelle und der eingesetzten Authentifizierungs-Proxies gehört. Gleichzeitig werden mehrere Gegenmaßnahmen vorgeschlagen, die genutzt werden können, wenn die Verbindungstabelle(n) einen einstellbaren Füllungsgrad erreichen. Darunter ist es möglich:

- die Größe der Tabelle dynamisch anzupassen,
- die am längsten inaktiven Verbindungen zu schließen,
- die Timeouts der eingetragenen Verbindungen bzw. die initialen Werte für den Verbindungsaufbau dynamisch anzupassen und zu skalieren (aggressive aging),
- TCP-Verbindungsversuche nicht mehr zwischenzuspeichern und SYN-Cookies[1] einzusetzen.

Die Abwendung einer übermäßigen Auslastung kann mit einstellbaren Grenzwerten für die jeweils maximale Anzahl der zwischengespeicherten IP-Fragmente, die maximalen Verbindungsversuche pro Quelladresse, die maximale Gesamtanzahl halboffener TCP-Verbindungen oder die maximale Anzahl von Log-Einträgen – das jeweils mit der Begrenzung pro Zeiteinheit – kombiniert werden. Mechanismen zur Limitierung der verschiedenen Ereignisse bieten alle Firewalls an.

[1] Bei SYN-Cookies werden Antwortpakete für den 3-Wege-Handshake so erstellt, dass kein Zustand gespeichert werden muss, aber eine legitime Antwort vom Initiator eindeutig erkannt werden kann.

Eine Möglichkeit Angriffe auf die Firewall zu erschweren ist, diese erst gar nicht als Vermittlungspunkt sichtbar zu machen. Dies kann als Nebeneffekt vom Betrieb der Firewall im Bridgemodus erreicht werden, wodurch die Firewall transparent für das Netzwerk arbeitet und nicht adressierbar ist. Im Routingmodus kann die Firewall Verschleierungsmethoden nutzen, die keine Informationen über sie oder ihren Zustand nach außen dringen lassen. Dies wird erreicht, indem die Firewall entweder keine Antworten und Fehlernachrichten sendet oder sich bei diesen als das Zielsystem ausgibt. Weiterhin kann sie beim Weiterleiten der IP-Pakete den TTL-Wert nicht wie üblich vermindern, wodurch sie als Weiterleitungsstation nicht sichtbar wird. Eine entsprechende Manipulation der TTL ist mit PF, IPFilter und iptables möglich. IPFW kann sich beim Ablehnen der Verbindungen als das Zielsystem ausgeben.

Für eine besonders hohe Verfügbarkeit bietet es sich an, die Firewall redundant auszulegen und die Zustandsdaten der einzelnen Einheiten untereinander zu synchronisieren. Die meisten Anbieter bieten eine Infrastruktur an, bei der bei einem Ausfall eine andere Einheit den Betrieb übernehmen kann. Aus lizenztechnischen Gründen unterscheiden die kommerziellen Anbieter oft zwischen mehreren vollwertigen und gleichzeitig einsetzbaren Einheiten im so genannten active/active-Betrieb und zusätzlichen passiven Failover- oder Standby-Einheiten, die erst beim Ausfall der aktiven Einheit den vollwertigen Betrieb aufnehmen können.

Die Verfügbarkeit von Netzwerkdiensten muss nicht unbedingt durch Angriffe oder den Ausfall von Soft- und Hardware gefährdet werden. Auch eine ungünstige Auslastung des Netzwerkes kann dazu führen, dass Daten nicht schnell genug ausgeliefert werden und Verzögerungen oder Ausfälle entstehen. Ebenso kann der Transport der Daten schneller sein als die Verarbeitung durch den Server. Hier helfen Techniken zur Kontrolle der Server- und Netzwerkauslastung, die die Verfügbarkeit von Diensten verbessern können. Für diese Aufgaben existieren spezialisierte Geräte (Rate Limiter, Traffic Shaper, Load Balancer), sie werden aber auch von Firewalls zur Verfügung gestellt.

Bandbreitenmanagement, Ratenbeschränkung und Traffic Shaping sind solche Techniken. Mit ihnen kann einem Dienst oder einem System eine untere bzw. eine obere Grenze, also die mindestens zugesicherte bzw. die maximal erlaubte Datenrate, für die Nutzung des Netzwerks zugewiesen werden. Diese Funktionen können sowohl als Sicherheits- wie auch als Netzwerkfunktionen betrachtet werden. Hier werden sie unter der ersten Kategorie vorgestellt, sollen aber unter beiden Kategorien eingeordnet werden.

Server load balancing sind Strategien, nach denen Anfragen nach bestimmten Kriterien an mehrere gleichartige Server verteilt werden, um die Auslastung der Dienste zu optimieren und die Verfügbarkeit zu gewährleisten.

Mit Quality of Service (QoS) können Datenpakete einer Dienstklasse und Weiterleitungsrichtlinie zugeordnet werden, die anderen vermittelnden Gegenstellen mitteilt, wie vorrangig sie behandelt werden sollen. In IP-Paketen ist dafür das Differen-

[2] Mehrere Load Balancing Lösungen sind unter `http://kb.linuxvirtualserver.org/wiki/Load_balancing` und `http://www.inlab.de/balance.html` gelistet.

System	Komponente(n)	traffic shaping	load balancing	QoS
netfilter	iproute2, u.a.[2]	✓	✓	✓
IPFW	dummynet	✓	?	?
IPF	altq, rdr nat	✓	✓	✓
PF	altq, rdr nat	✓	✓	✓
PIX	integriert	–	–	ab ver. 7.0
FW-1	floodgate	✓	?	✓

Tabelle 3.1: Unterstützte traffic Regulierungen

tiated Services Code Point (DSCP) bzw. Type of Service (TOS) Feld vorgesehen. Unterstützende Firewalls können darauf basierend filtern, die Markierung verändern oder priorisiert weiterleiten.

Tabelle 3.1 gibt eine Übersicht über die Unterstützung der einzelnen Mechanismen durch die Firewalls und benennt die zuständigen Komponenten dazu.

AAA – Authentication, Authorization, Accounting

Bei der Authentifizierung von Benutzern gegenüber der Firewall haben sich in erster Linie die Protokolle RADIUS, TACACS+ und zunehmend auch Kerberos etabliert. Zusätzlich bieten manche Anbieter eine Authentifizierung über eine gesicherte Weboberfläche oder verlangen eine vorherige Authentifizierung über ssh bzw. inzwischen seltener über telnet. Wird eine Web-Proxy-Funktion mit angeboten, so ist eine Authentifizierung auch über das Proxy-Protokoll SOCKS V5 möglich.

Netfilter bietet als Framework selbst keinen Authentifizierungsdienst an, kann aber mit verschiedenen externen Projekten ergänzt werden. Darunter wäre z.B. das NuFW-Projekt[3] zu nennen, das eine differenzierte Benutzerauthentifizierung über ein client-Programm auch auf Mehrbenutzersystemen erlaubt.

IP FireWall und IPFilter bieten ebenfalls eine allgemeine Schnittstelle an, um Programme zur Authentifizierung von Benutzern anzubinden. Bei IPFW werden dazu *divert*-, bei IPF *auth*-Regeln benutzt. Ein konkreter Dienst wird darüber allerdings nicht angeboten. In beiden Fällen müssen nach der Authentifizierung zusätzlich entsprechende Regeln für folgende Pakete gesetzt werden. IPF erleichtert diese Aufgabe durch spezielle *preauth*-Regeln, die auf eine dynamische Liste mit Authentifizierungsregeln zurückgreifen können. Wird dort eine weitere passende Regel gefunden, wird ihre Filterentscheidung benutzt. Andernfalls wird das Paket geblockt.

PF bietet einen vollständigen Dienst über authpf an. Hierbei können sich Benutzer über SSH auf der Firewall anmelden, woraufhin ihre Quelladresse für die Dauer der SSH-Verbindung für konfigurierte Verbindungen frei geschaltet wird.

Die Cisco PIX kann über die Protokolle RADIUS und TACACS+ neben Authentifizierungs-, auch Autorisierungs- und Accounting-Dienste von einem separaten Server durchleiten bzw. benutzen.

[3] `http://www.nufw.org/`

Checkpoint FW-1 unterscheidet zwischen Benutzer-, Client-, und Session-Authentifizierung und bietet dafür jeweils unterschiedliche Methoden an. Für eine detaillierte Aufschlüsselung wird auf das Handbuch verwiesen.

Vertraulichkeit und Integrität

Häufig werden die AAA-Funktionen mit Virtual Private Network (VPN) Diensten und kryptographischen Verfahren verknüpft. Verschlüsselung kann einer erhöhte Form der Authentifikation unterstützen, Vertraulichkeit gewährleisten und Integrität der Daten sicherstellen.

Die einzelnen Produkte unterscheiden sich in den unterstützten VPN-Techniken, den Betriebsmodi, der maximalen Anzahl gleichzeitiger Verbindungen oder dem zugesicherten verschlüsselten Datendurchsatz. Bei den Betriebsmodi handelt es sich um die Anzahl der Endpunkte auf beiden Seiten des gesicherten Tunnels: $n : m$ bzw. Site-to-Site, $1 : 1$ alias End-to-End und $1 : n$ im so genannten Roadwarrior-Szenario. Je nach verwendetem VPN-Verfahren werden unterschiedliche Modi unterstützt. VPNs können auf mehreren OSI-Schichten realisiert werden. Auf Schicht 2 kann PPP mit Verschlüsselung, PPTP und L2TP eingesetzt werden. IPSec ist der bekannteste Vertreter der Schicht 3. Auf Anwendungsebene können Tunnel mit SSL bzw. TLS oder mit SSH aufgebaut werden.

Vernetzungsmerkmale

Die kommerziellen Systeme unterscheiden sich in der Schwerpunktsetzung auf entweder Routing mit Firewallfähigkeiten oder Firewall mit Routingfähigkeiten[4].

Neben statischem IP-Routing beherrschen die untersuchten Firewallsysteme auch dynamische Routingtabellen mittels der Routingprotokolle OSPF, BGP oder RIP. Besondere Routingvarianten wie policy-based routing (PBR) – also das Weiterleiten von Paketen basierend auf weiteren Merkmalen wie Quelladresse oder Zielport – oder redundante Routen, die gewählt werden, wenn die primäre Verbindung nicht verfügbar ist, sind eingeschränkt bei manchen Produkten verfügbar.

NAT wird von allen Systemen unterstützt – die Anzahl der verfügbaren Strategien bei dynamischem NAT unterscheidet sich jedoch. Weiterhin sind die unterstützten NAT-Arten für Anwendungsprotokolle ein Unterscheidungsmerkmal. IPFW benötigt für NAT bei FreeBSD Versionen bis einschließlich 6.2 ein separates Programm (natd), das an jeder beliebigen Stelle im Regelwerk über die *divert*-Schnittstelle aufgerufen werden kann. In der Version 7.0 wurde die NAT-Unterstützung im Kernel realisiert.

Bei IPFilter wird NAT ebenfalls über ein gesondertes Regelwerk und das Kommandozeilenprogramm ipnat gesteuert. Dabei unterstützt es neben source und destination NAT auch ausgefallenere Varianten von dynamischem NAPT, bei dem auch individuelle Portbereiche, ICMP ID Übersetzung, ein Round-Robin-Verfahren bzw.

[4] Cisco IOS Router können um ein Firewall-Modul erweitert werden, wodurch sie auch komplexere Filterungen durchführen können. Die Cisco PIX Firewall hat im Gegenzug auch eingeschränkte Routingfunktionen.

einfaches Load Balancing angegeben werden können. Für die NAT-Umsetzung von Protokollen auf Anwendungsebene (z.B. FTP) können Proxies im Kernel oder transparente Proxies im Userspace eingesetzt werden.

In Abschnitt 2.1 auf Seite 7 wurden die zwei Betriebsarten Routing- und Bridgemodus vorgestellt. Außer der Cisco PIX[5] unterstützen alle Firewallsysteme beide Betriebsarten. Im Bridgemodus wie auch im Routingmodus mit dem gleichzeitigen Einsatz von SNAT nimmt die Firewall die Pakete für die hinter ihr liegenden Systeme entgegen, indem sie sich für sie ausgibt. Dies wird erreicht durch das Aktivieren von ProxyARP auf den gewünschten Netzwerkschnittstellen. ProxyARP kann bei allen Systemen konfiguriert werden.

Eine zusätzliche Separationsebene zwischen den Systemen in einem Netzwerk kann mittels VLAN erfolgen, das ein physisches Netz in weitere logische und voneinander getrennte Netzwerke aufteilt. Dadurch kann einerseits die Sicherheit erhöht werden, da nur noch Systeme innerhalb eines logischen Netzes miteinander kommunizieren können, und andererseits wird durch die gleichzeitige Einschränkung der Broadcast-Domänen die Netzwerkauslastung verbessert. Alle untersuchten Systeme unterstützen den Betrieb von VLANs.

3.2. Merkmale der Regelverarbeitungs- und Filtereinheiten

Bei der Untersuchung der Firewallsysteme wurden unterschiedliche Abbildungen des Regelwerks, dessen Abarbeitung und Voreinstellungen identifiziert. Ziel der folgenden Darstellung ist die Vorstellung der Besonderheiten und der Mächtigkeit der jewiligen Konfigurationssprachen. Durch die Kategorisierung der verfügbaren Sprachbefehle soll zudem eine Trennung zwischen den Strukturierungsmitteln der Sprache und den Anweisungen für den Paketfilter erleichtert werden. Letztere sind für die Bewertung der Mächtigkeit und der Testbarkeit eines Paketfilters relevant.

Bei allen untersuchten Firewalls ist die Syntax der Konfigurationssprachen dokumentiert. Bei der PF und bei IPF ist sie sogar in Backus Naur Form (BNF) spezifiziert, wodurch Uneindeutigkeiten verhindert werden[6]. Die Semantik, die genauen Auswirkungen einer Einstellung und die Interpretation von möglichen Ausgaben werden aber oft nur in Kurzform gehalten. Das Handbuch der Cisco PIX ist ein typisches Beispiel, in dem Befehlsoptionen häufig nur in Satzform benannt, die Auswirkungen jedoch nicht erläutert werden. Fehlende, uneindeutige oder widersprüchliche Dokumentation führt dazu, dass der Benutzer raten muss. Die Überprüfung oder Ergänzung der Dokumentation liefert eine Motivation für diese Arbeit und für das Testen von Firewalls.

[5] Cisco PIX unterstützt den Bridgemodus ab der Softwareversion 7.0, die hier nicht betrachtet wird.

[6] Zur Konfigurationssprache der PF vgl. die manpage `pf.conf(5)`. IPFilter ist in [CF02] und in der manpage `ipf(5)` dokumentiert.

Struktur des Regelwerks

Alle untersuchten Firewalls verwalten die Regeln in Listen oder Tabellen. Bei der FW-1 sind sie jedoch nicht sichtbar, da es nur eine grafische Ein- und Ausgabemöglichkeit gibt[7]. Üblicherweise werden die Regeln sequentiell nach dem *first-match*-Prinzip bearbeitet – Ausnahmen sind IPF und PF, die standardmäßig im *last-match*-Modus arbeiten, aber auch regelweise umgeschaltet werden können. Diese Möglichkeit führt zwar zu einer höheren Mächtigkeit der Konfigurationssprache, da die Abarbeitung der Regeln vorzeitig abgebrochen werden kann, macht aber gleichzeitig die Benutzerschnittstelle im Mischbetrieb fehleranfälliger und vermindert die Benutzbarkeit.

Bei iptables gibt es fünf vorgegebene Zugriffspunkte auf Pakete im Paketfluss, an denen der Benutzer seine Regeln in Abarbeitungsketten (*chains*) definieren kann. Der Benutzer kann die Systemketten mit eigenen Ketten erweitern und in sie verzweigen. Das Aufrufen weiterer Ketten kann bei übereinstimmenden Kriterien wahlweise die aktuell bearbeitete Kette ersetzen (*–jump*) oder erweitern (*–goto*). Die vorgegebene Aktion RETURN erlaubt einen Rücksprung in der Abarbeitungskette und setzt die Ausführung in der letzten Kette, aus der mit *jump* verzweigt wurde, fort. Für die vorgegebenen Systemketten kann eine Standardentscheidung eingestellt werden, die sich auch auf die jeweils verknüpften benutzerdefinierten Ketten auswirkt.

IP FireWall arbeitet mit 32 durchnummerierten Sets, die jeweils 65535 Regelblöcke unterscheiden können, bietet dafür aber nur ein gemeinsames Regelwerk für alle Flussrichtungen an. Mehrere Regeln können pro Regelnummer angegeben werden. Sie werden dann sequentiell in der Reihenfolge ihrer Erstellung abgearbeitet. Über das Schlüsselwort *skipto* kann der Benutzer zu einer weiter vorne liegenden Regelnummmer verzweigen, wodurch die Abarbeitung von unzutreffenden Regeln umgangen werden kann. Sets können manuell ein- und ausgeschaltet werden. Die Regelnummer 65535 im letzten Set übernimmt die Rolle einer default policy.

Die Konfiguration des IPFilters erlaubt es gleichzeitig eine aktive und eine inaktive Regelliste anzulegen, was die unterbrechungsfreie Pflege bzw. den atomaren Austausch des Regelwerks ermöglicht. Eine Besonderheit bei IPF ist die Reihenfolge der Regelbearbeitung, die von vorne nach hinten durchläuft, wobei die letzte passende Regel angewendet wird. Über das Schlüsselwort *quick* kann bei Übereinstimmung der Bedingungen die Anwendung der aktuellen Regel erzwungen werden. Regeln können auf mehreren Ebenen hierarchisch gruppiert werden, indem die einer Gruppe zugeordneten Regeln (Schlüsselwort *group*) über eine Kopfregel (*head*) angesprungen werden. Zur weiteren Strukturierung können über das Schlüsselwort *skip*, gefolgt von der Anzahl der Regeln, Regeln gezielt übersprungen werden.

Die PF Firewall arbeitet, wie die IPF, ebenfalls nach dem *last-match*-Prinzip und kann über das Schlüsselwort *quick* die Abarbeitung terminieren. Zur Strukturierung des Regelwerks können Ankerpunkte (*anchors*) definiert werden, die als Unterbaum abgearbeitet werden. Diese können ähnlich wie Verzeichnisse im Dateisystem (z.B.

[7] Ein externes Projekt, das die Konfiguration einlesen und wiedergeben kann ist unter `http://www.wormnet.nl/cprules/` zu finden.

`/incoming/mail/spamfilter`) über mehrere Ebenen angelegt werden und dabei auch in übergeordnete Teilbäume (z.B. `../proxy`) zurückspringen.

Alle Cisco Produkte, so auch die PIX, arbeiten mit so genannten access-lists. Diese werden sowohl für die Filterung wie auch für policy NAT und die Zuordnung der VPN-Tunnel benutzt. Access-lists gelten grundsätzlich für den ankommenden Verkehr und werden an eine Netzwerkschnittstelle, eine NAT- oder eine VPN-Definition gebunden. Sie können nicht weiter strukturiert werden.

Checkpoint Produkte werden über das grafische Programm SmartDashboard administriert. Dort werden alle Objekte in statischen und dynamischen Gruppen gepflegt, auf die sich das Regelwerk bezieht. Es gibt je eine Regelliste für Filterung, NAT, VPN, Quality of Service und weitere installierte Zusatzkomponenten. Für jeden Eintrag in der Liste werden die Zielsysteme definiert, auf denen diese Regeln installiert und durchgesetzt werden. Die interne Abbildung des Regelwerks konnte nicht ermittelt werden.

Die Firewalls IPFW und PIX erstellen auch dynamisch neben dem vom Benutzer vorgegebenen Regelwerk Regeln. IPFW erstellt Regeln als Repräsentation der Zustandsinformationen. Die Cisco PIX erstellt dynamisch Regeln für eingehende VPN-Verbindungen. Sie erlaubt auch eine bedingte Reglementierung über den Befehl *established*, der Assoziationen in Abhängigkeit von einer anderen bereits bestehenden Assoziation erlauben kann.

Objektgruppen

Bis auf IPFilter bieten alle untersuchten Firewalls eine Möglichkeit Regeln zusammenzufassen, indem sie sich nicht auf einzelne Eigenschaften bzw. Objekte, sondern auf Objektgruppen beziehen.

Linux netfilter kann besonders schnell auf Gruppen von IP-Adressen und Ports zugreifen, die mit dem Werkzeug ipset verwaltet werden.

IP FireWall bietet gleich mehrere Möglichkeiten an, Objektgruppen zu verwalten. Mit *or-blocks* (Bsp. { `x or not y or z` }) lassen sich Alternativen zusammenfassen. Adressen können mit *Address Sets* zusammengefasst (Bsp. `1.2.3.4/24{128, 35-55, 89}`) oder in *Lookup Tables* als ein Schlüssel-Wert-Paar gespeichert werden. Der Schlüssel wird durch IP-Adresse/Maske gebildet und der Wert kann als Parameter für weitere Regeln benutzt werden. Or-blocks und Address Sets lassen sich in Variablen ablegen und in den Regeln referenzieren.

PF kann, ähnlich zur IPFW, Adressen und Ports in Listen zusammenfassen (Bsp. `{10.1.0.0/24, 10.2.1.1}`), sie über Variablen referenzieren sowie Tabellen benutzen. Ein besonderer Anwendungsfall für Tabellen ist das dynamische Hinzufügen von Elementen beim Überschreiten von definierten Obergrenzen für Verbindungen oder Verbindungsversuche. Die betroffenen Einträge (z.B. angreifende Systeme) können dann mit anderen Regeln gesondert behandelt werden. Vergleichbare Funktionalität ist bei iptables mit dem *recent*-Modul erreichbar, das nach frei wählbaren Kriterien Quelladressen in Tabellen eintragen bzw. aktualisieren kann.

Die Objektgruppen der PIX können Netzwerke, Systeme, Dienste, Protokolle,

ICMP-Typen und weitere Objektgruppen vom gleichen Typ zusammenfassen. Sie werden anschließend in den access-lists referenziert.

Die gesamte Konfiguration der FW-1 basiert auf Objekten. Vor der Regelerstellung muss der Administrator alle Netzwerkobjekte anlegen und kann sie daraufhin in den Regeln referenzieren. Dienste sind bereits vordefiniert und können erweitert werden.

Eine weitere Möglichkeit Eigenschaftsbereiche zusammenzufassen wird vor allem in NAT-Regeln verwendet und ist deshalb oft unter der Bezeichnung NAT-Exemption anzutreffen. Hierbei handelt es sich um die Möglichkeit, zuerst spezielle Bereiche aus dem NAT auszuschließen, bevor eine allgemeine Regel das Standardverhalten definiert. Solch eine Möglichkeit bieten der Packet Filter über *no nat*, *no rdr* und *no binat* sowie Cisco PIX über *nat 0*. Bei iptables lässt sich ähnliche Funktionalität durch Verzweigungen in den Regellisten mit *jump* und *goto* erreichen. Bei IP FireWall und IPFilter konnte für NAT kein Ausschlussmechanismus gefunden werden. FW-1 konnte nicht überprüft werden.

Mächtigkeit des Paketfilters und der Paketveränderungen

Ein Unterschied zwischen den untersuchten Open Source und den kommerziellen Paketfiltern sind voreingestellte Konsistenzprüfungen des Datenverkehrs und Schutzfunktionen, die in den kommerziellen Systemen oft als implizite Regeln oder sogar Teil des Paketfilters bzw. IDS implementiert sind. In den freien Paketfiltern können ähnliche Funktionalitäten erreicht werden, müssen aber bei der Konfiguration u.U. manuell eingestellt werden. Dies soll nicht unbedingt als Nachteil, sondern als Flexibilität der Einstellungsmöglichkeiten eines Paketfilters betrachtet werden.

Die verfügbaren Filterkriterien sind ein Indiz dafür, wie genau der Benutzer durch Einstellung der Regeln die Inspizierung der Pakete oder der Paketflüsse kontrollieren kann, um Entscheidungen zu treffen oder Modifikationen der Pakete durchzuführen.

In den betrachteten kommerziellen Paketfiltern wie auch in den meisten Publikationen zur zustandsbehafteten Filterung werden vor allem die Kriterien *src, dst, sport, dport, interface* und im Falle von TCP die Flags als Repräsentanten für den Zustand berücksichtigt. Andere Kriterien werden ignoriert oder zur Verringerung der Komplexität vor dem Benutzer versteckt. Dabei können auch alle anderen Paketmerkmale wie z.B. die IP-TTL, die IP- und TCP-Optionen, die Paketlänge, die Fragmentierung, die QoS Flags, der Payload oder die Datenrate entscheidend für eine feingranulare Reglementierung sein. Als besondere Filterkriterien unterstützen manche Paketfilter die Tageszeit (FW-1), eine Betriebssystemerkennung (PF) sowie eine Trefferquote im Sinne von „jedes n-te Paket“ oder „im Schnitt n%“ (iptables, PF). Nur netfilter/iptables kann sich direkt auf den Zustand der Assoziation, der nicht nur für TCP angelegt wird, beziehen und diesen in den Regeln verwenden. Es kann noch mit vielen weiteren (in)offiziellen match-Kriterien ergänzt werden.

Alle Firewallsysteme bieten die Möglichkeit an, IP-Adressfälschungen (IP Spoofing) zu verhindern. Dafür wird bei den eingehenden Paketen die Quelladresse und die eingehende Netzwerkschnittstelle mit der Routingtabelle verglichen, um zu ermitteln, ob eventuelle Antwortpakete den gleichen Weg nehmen würden (source-route-verify).

Gleichzeitig wird sicher gestellt, dass die Quelladressen von ausgehenden Paketen entweder der Firewall oder einem System aus den angeschlossenen Netzwerken zugeordnet werden können. Diese Überprüfungen müssen entweder manuell konfiguriert werden, sind in einer Option zusammengefasst oder können als eine Regel angegeben werden.

Um Missbrauch zu verhindern, können auf TCP zahlreiche Überprüfungen angewendet werden. Durch strikte Überprüfung der im Paket gesetzten TCP-Flags und der aktuellen Phase der Verbindung können viele Portscanarten (z.B. Window-Stealth oder Null-, Xmas-, Fin- und ACK-Scans) geblockt werden. Kombiniert mit einer Begrenzung maximaler Verbindungsversuche pro Quelladresse können z.B. TCP half-open und full-connect Scans verlangsamt werden. Die Art und Weise der zustandsbehafteten Untersuchung der Datenströme durch die Firewall ist ein Gegenstand der vorliegenden Untersuchung. Die Überprüfungen lassen sich nur bei iptables vom Benutzer einstellen, da die anderen Systeme das automatisch übernehmen.

Bei manchen Paketfiltern lassen sich neben der Adressumsetzung auch weitere Veränderungen an den Paket-Headern der Internet- und Transport-Schicht durchführen bzw. einstellen. Um Systeme mit vorhersagbaren Identifikations- oder Sequenznummern zu schützen, können die TCP-Sequenznummer, die Identifikationsnummern in IP-, ICMP- und DNS-Paketen sowie der IP- und TCP- Zeitstempel mit einem zufälligen Offset modifiziert werden. Dadurch kann die Erkundung der Merkmale der geschützten Systeme erschwert werden. Viele Angriffe nutzen Fehlinterpretation von IP-Fragmenten aus, weswegen der Paketfilter diese verwerfen oder neu zusammenstellen und Uneindeutigkeiten[8] bereinigen kann.

Ebenfalls zu den Paketveränderungen gehört das Entfernen von optionalen Daten. Besonders häufig wird das bei IP-Optionen gemacht, die zwar von der Spezifikation vorgesehen wurden, aber aus Sicherheitsgründen bedenklich sind. Sie können von manchen fehlerhaften Netzwerkstapeln nicht interpretiert werden und lassen sich für verschiedene Angriffe missbrauchen. So werden von den meisten Firewalls Möglichkeiten geboten zumindest die IP-Routing-Optionen zu filtern bzw. zu verwerfen.

Die mächtigsten Veränderungen ermöglichen PF und netfilter, wobei PF mit den *scrub*- bzw. *modulate*-Optionen mehrere sicherheitsrelevante Modifikationen zusammenfasst und netfilter für jedes Paketfeld eine spezialisierte und frei konfigurierbare Option anbietet. IPFilter und IPFireWall bieten keine Möglichkeiten an, Pakete zu modifizieren. Pakete könnten aber an Benutzerprogramme übergeben werden, die evtl. Veränderungen durchführen könnten. Solche Erweiterungen sind jedoch zur Zeit nicht bekannt. Die beiden kommerziellen Produkte Cisco PIX und Checkpoint FW-1 bieten ebenfalls keine direkten Möglichkeiten an, Paketfelder zu verändern. Bestimmte Änderungen lassen sich jedoch über Optionen einschalten. Dazu gehören vor allem die Randomisierung der Sequenznummer und Zeitstempel sowie die Entfernung von

[8] In der Regel sind doppelte Fragmente, Fragmente außerhalb der Reihenfolge und welche mit überlappenden Bereichen Anzeichen für einen Angriff. Manchmal ist auch die wiedersprüchliche Dont-Fragment-Markierung im IP-Header von Fragmenten gesetzt. Manche nicht standarkonformen IP-Stacks verschicken jedoch unter bestimmten Umständen solche Fragmente.

IP-Optionen. Jede der Firewalls bietet Möglichkeiten an, IP-Spoofing (mehr oder weniger komfortabel) zu verhindern.

3.3. Vergleich und Bewertung der Paketfilter

Die Mächtigkeit der Paketfilter in den untersuchten Firewallprodukten ist vergleichbar, auch wenn nicht bei allen die Abläufe im Paketfilter, aufgrund der unvollständigen Dokumentation und Monitoringmöglichkeiten, sichtbar werden oder vom Benutzer einstellbar sind. Die Unterschiede liegen vor allem in der Integration der verschiedenen Merkmale in ein für die Aufgabe möglichst optimiertes Gerät und die dazu passende Software. Dazu kommen noch Werkzeuge, die dem Benutzer die Konfiguration, die Überprüfung der Funktionaliät sowie die Diagnose ermöglichen und vereinfachen. Die Funktionsunterschiede ergeben sich aus den internen Designentscheidungen der gewählten Firewall-Architektur, den durch die Hardware vorgegebenen Leistungsmerkmalen sowie künstlichen Beschränkungen, um verschiedene Lizenzen bzw. Produkte anbieten zu können. Bei weniger integrierten Lösungen, von denen eine höhere Spezialisierung oder Leistung erwartet werden kann, müssen die fehlenden Funktionen durch weitere Komponenten bereitgestellt werden.

Durch einen Vergleich der Funktionalitäten der untersuchten Paketfilter konnte netfilter/iptables als das substanzielle System mit der größten Konfigurationsfreiheit ermittelt werden. Tabelle 3.2 zeigt die verfügbaren Filterkriterien von iptables, die für die möglichen Entscheidungen und Aktionen aus Tabelle 3.3 herangezogen werden können. Bis auf die Aktionen, die unter „nicht unterstützt“ gelistet sind, können alle Funktionen bzw. Optionen der vorgestellten Paketfilter als gleichwertige iptables-Regeln formuliert werden. Als gleichwertig werden die extern beobachtbaren Resultate und nicht unbedingt die internen Mechanismen bzw. Strategien verstanden.

Protokoll und Protokollfelder	
Layer 3	IP, IP-Fragmente, TTL, TOS/DSCP/ECN, IP-Optionen
Layer 4	TCP, UDP, ICMP, IPSEC, DCCP, SCTP
Adressierung	
Layer 1	interface (in, out, physical)
Layer 2	MAC, pkttype (ucast, bcast, mcast)
Layer 3	IP (iprange, ipset), addrtype (ucast, bcast, mcast, local, anycast, blackhole, unreachable, prohibit, throw, nat, xresolve, unspec)
Layer 4	ports
Layer 5	BGP routing realm
Limits	
connlimit	connections/source (TCP only)
hashlimit	pkts/sec (Hash über Liste aus: srcip, srcport, dstip, dstport)
limit	pkts/t, t=sec,min,hour,days
fuzzy	pkts/sec
quota	pkts max
connrate	bytes/sec
connbytes	mode/direction (mode: pkts, bytes, avgpkt; direction: in/out/both)

Paket	
Paket	length, string/hexstring, unclean, fragmented
Zustand	
Verbindung	Association state (NEW, EST, REL, INV, UNTRACKED), TCP state, Ebene der Assoziation (childlevel)
Markierungen	
Paket	mark
Verbindung	connmark
Spezialkriterien	
Zustand	recent, condition from file, Port-Scan-Detector
Andere	n^{th}, random, OS fingerprint, owner, time

Tabelle 3.2: Filterkriterien von iptables

Filterentscheidung	
passiv	ACCEPT, DROP
aktiv	REJECT, tarpit (Verlangsamt TCP-basierte Spoofing-Angriffe)
delegiert	nfqueue (Weitergabe an Benutzerprozess, z.B. IDS), transparent proxy (inoff. Erweiterung)
Reroute	
1 Ziel	DNAT, SAME
n Ziele	balance DNAT, ClusterIP, Route Paket oder Kopie
lokal	local redirect, transparent proxy
Paketmodifikationen	
Adressen	SNAT, DNAT, MASQ (sowohl im IP-, TCP- und UDP-Header als auch für Anwendungsprotokolle)
IP und TCP	TTL, IP-Optionen, TCP MSS
Markierungen	
Paket	TOS/DSCP/ECN
Paketbeschreibung	IP mark, CLASSIFY CBQ, mark, notrack
Verbindungseintrag	connection mark
Gruppen	ipset
Logging und Accounting	
	log, ulog, account
nicht unterstützt	
Modifikationen	DF/MF-Flags, ID (IP, DNS, ...), TCP-Sequenznr., IP- und TCP-Timestamp
Anti-DoS	SYN-Proxy

Tabelle 3.3: Mögliche Aktionen des iptables Paketfilters

4. Funktionsweise der Paketfilter

In diesem Kapitel wird die Analyse der betrachteten Firewalls fortgesetzt. Diesmal stehen nicht mehr die Merkmale, sondern die Funktionsweise und die internen Abläufe im Vordergrund der Betrachtung.

In Abschnitt 4.1 werden die Paketflüsse der Paketfilter betrachtet. Sie sollen dazu genutzt werden die Zugriffs- und die Entscheidungspunkte des Paketfilter sowie die Bearbeitungsschritte für die empfangenen, die weitergeleiteten und die gesendeten Pakete zu identifizieren. Zusätzlich soll die Reihenfolge und die Abhängigkeiten zwischen den bereits ermittelten Merkmale und Funktionen festgehalten werden. Beide Aspekte können dazu beitragen die Tests genauer zu steuern und sie einzustellen.

Im zweiten Abschnitt wird die zustandsbehaftete Verbindungsverfolgung vorgestellt, die ein Mechanismus der Firewalls ist, mit dem die Zustände der Endpunkte verfolgt und verwaltet werden. Die Analyse dieses Mechanismus ist notwendig, um eine Grundlage zu schaffen, die von dem Paketfilter unterschiedenen Zustände bei der Teststrategie zu berücksichtigen und sie gezielt ansteuern zu können.

4.1. Paketflüsse

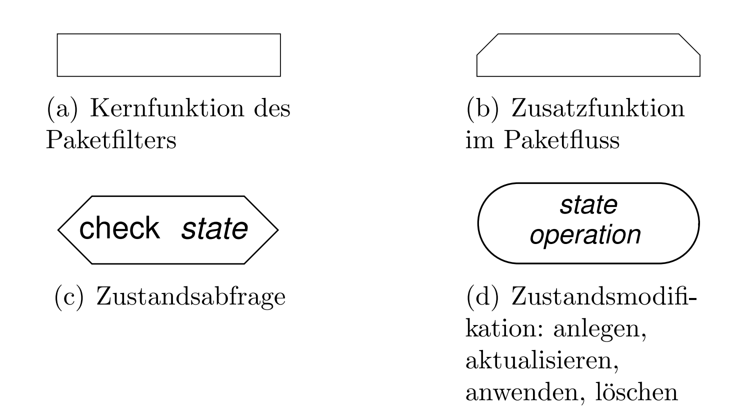
Die vorgestellten Funktionen der Firewalls bzw. der Paketfilter werden entwurfsbedingt von den einzelnen Probanden in unterschiedlichen Reihenfolgen abgearbeitet. Diese Reihenfolge bestimmt die Prioritäten der Sicherheitsfunktionen und ist einerseits ein Mittel zur Entlastung der Firewall von komplexeren Überprüfungen, wenn ein Paket bereits aufgrund der Nichtübereinstimmung mit einfacheren Kriterien verworfen werden kann, und andererseits bestimmt sie die Einflussnahme der früheren Mechanismen auf die darauf folgenden.

In diesem Abschnitt wurden die sechs ausgesuchten Firewalls – netfilter/iptables, IP FireWall, IPFilter, Packet Filter, PIX und FireWall-1 – anhand der Dokumentation und sofern vorhanden anhand des Quellcodes analysiert. Die identifizierten Paketflüsse werden beschrieben und grafisch repräsentiert. Für die Darstellung der Funktionsblöcke werden in Abbildung 4.1 Symbole eingeführt, die für die unterschiedenen Funktionstypen verwendet werden.

netfilter/iptables

Die Grundidee des netfilter-Frameworks basiert auf der Einführung von fünf Zugriffspunkten (*hooks*) in verschiedenen Phasen der Verarbeitung von Paketen im Netzwerkstack, die in Abbildung 4.2 zu sehen sind.[1] Ein vom Netzwerkadapter oder

[1] Für eine ausführliche Betrachtung des netfilter-Frameworks siehe [Ayu06; Spe06; RW02].



(a) Kernfunktion des Paketfilters

(b) Zusatzfunktion im Paketfluss

(c) Zustandsabfrage

(d) Zustandsmodifikation: anlegen, aktualisieren, anwenden, löschen

Abbildung 4.1: Symbole für Kern- und Zusatzfunktionen im Paketfluss

vom lokalen System kommendes Paket kann somit einen von drei möglichen Wegen durch die Netzwerkbehandlungsroutinen nehmen. Ein ankommendes Paket betritt das System über `PREROUTING`, wonach die routing-Entscheidung getroffen wird. Ist das Paket für das lokale System bestimmt, so geht es weiter zum Punkt `INPUT`. Ein Paket, das weitergeleitet werden soll, geht weiter über `FORWARD` und `POSTROUTING`. Lokal auf dem System erstellte Pakete treffen über den Zugriffspunkt `OUTPUT` ein und gehen ebenfalls über das `POSTROUTING` weiter.

An jedem der fünf Zugriffspunkte können sich netfilter Erweiterungen mit callback-Funktionen unterschiedlicher Priorität registrieren, um über ein neues Paket informiert zu werden. Daraufhin verarbeiten sie das Paket und melden einen Verarbeitungsstatus zurück. Neben den Funktionen des Paketfilters passieren die Pakete weitere Behandlungsroutinen. Darunter sind die Quality-of-Service-Behandlung und das Routing.

Im netfilter-Framework, einschließlich der aktuellen Version in Linux 2.6.20, können die Erweiterungen unter folgenden Entscheidungen wählen: Paket wurde akzeptiert und darf weiter verarbeitet werden (ACCEPT), Paket wird verworfen (DROP), Paket wird im Userspace weiterverarbeitet (QUEUE), Weiterbehandlung wartet auf ein Ereignis und Paket wird festgehalten (STOLEN) sowie das Paket soll den Zugriffspunkt nochmal von vorne durchlaufen (REPEAT).

Das Konfigurationsprogramm iptables nutzt diese Infrastruktur und verwaltet die Benutzerregeln in mehreren Tabellen, die sich je nach Aufgabe an unterschiedlichen Zugriffspunkten registrieren. iptables in der untersuchten Version 1.3.5 kennt in der Standardkonfiguration die Tabellen *filter*, *nat*, *mangle*, *raw* und *conntrack*, wobei nur die ersten vier vom Benutzer manipuliert werden können.

filter Die Tabelle *filter* ist für die Paketfilterung zuständig und wird an den Zugriffspunkten `INPUT, FORWARD` und `OUTPUT` registriert.

nat In der Tabelle *nat* können Regeln zur Veränderung der Quell- und Zieladresse der Pakete angegeben werden. Sie wird an den Zugriffspunkten `PREROUTING, OUTPUT` und `POSTROUTING` registriert.

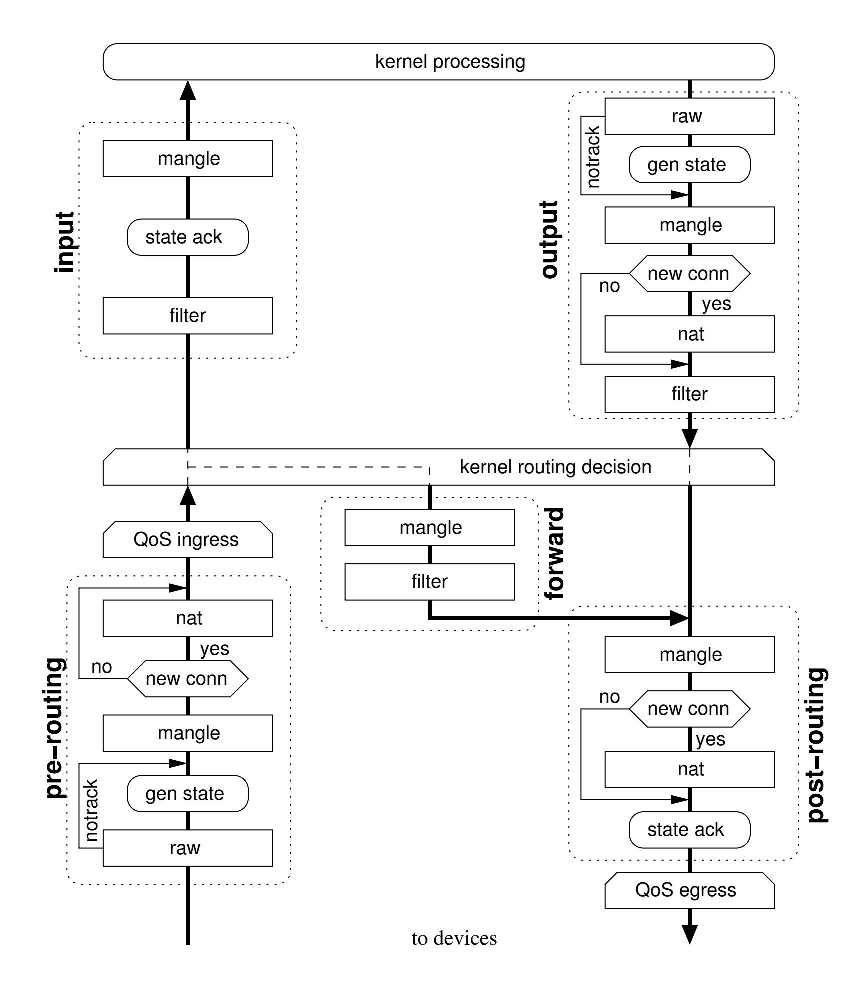


Abbildung 4.2: Entscheidungspunkte für den Paketfluss bei iptables

mangle In der Tabelle *mangle* können Änderungen an Inhalten und Markierungen der Pakete vorgenommen werden. Dazu zählen Veränderungen von QoS- und ToS-Feldern oder systeminterne Markierungen für das Bandbreitenmanagement. *mangle* wird an allen fünf Zugriffspunkten registriert.

conntrack Die Tabelle *conntrack* steht dem Benutzer nicht zur Verfügung und wird vom System intern benutzt, um zustandbehaftete Verbindungsverfolgung zu realisieren. Sie wird in `PREROUTING` und `OUTPUT` registriert, wo neue Zustände erstellt werden (in der Abbildung mit *add state* gekennzeichnet). In den Zugriffspunkten `POSTROUTING` bzw. `INPUT` werden die Einträge bestätigt (*state ack*). Diese Zweiteilung wird durchgeführt, um nur gefilterte und zugelassene Pakete zu verwalten.

raw Die *raw*-Tabelle wurde erst nachträglich (2003) hinzugefügt, um bestimmte Pakete von der Verbindungsverfolgung ausschließen zu können. Sie wird vor dem *conntrack* in `PREROUTING` und `OUTPUT` registriert.

Andere netfilter-bezogene Parameter, die bei der Konfiguration und vor allem bei

der Verbindungsverfolgung berücksichtigt werden sollten, steuern die Timeouts, die Größe der Tabelle oder das Logging. Ein Ausschnitt der wichtigsten Parameter ist im Anhang in Tabelle A.2 gelistet und in [Spe06] ausführlich beschrieben.

IPFW - IP FireWall

In dieser Sektion zu IP FireWall werden die zwei gefundenen Varianten des Paketfilters vorgestellt. Die erste Variante wurde unter FreeBSD analysiert und ist heute noch im Einsatz. Die Variante von BSD/OS ist nicht mehr verfügbar, wird aber aufgrund der abweichenden Architektur vorgestellt, die gleichzeitig als einzige mit der netfilter-Architektur vergleichbar ist.

Bei FreeBSD IPFW kann ein Paket an mehreren Stellen des Protokollstapels mit den aktiven Regellisten verglichen werden, was durch Systemvariablen gesteuert wird (vgl. dazu auch die Struktur des Regelwerks in Abschnitt 3.2). So variiert die Anzahl der Zugriffspunkte zwischen 0 und 4, abhängig von der Konfiguration (aktiviertes Bridging, Filterung von Ethernet, Filterung von IP) sowie der Quelle bzw. dem Ziel des Paketes. Jedes Paket wird jedesmal mit der gesamten aktiven Regelmenge verglichen, was bei der Formulierung der Zugriffskriterien berücksichtigt werden muss. Ebenso müssen die vom Benutzer frei wählbare Platzierung des Übergabepunktes zum NAT-Dienst und die folgenden Filterregeln bedacht werden.

Abhängig vom Ort des Zugriffs können der Regelliste unterschiedliche Informationen zur Verfügung stehen, da Kopfdaten je nach Netzwerkschicht entfernt oder hinzugefügt werden können. Deshalb wird der positive Vergleich von MAC- Adressen innerhalb von *ip_input()* bzw. *ip6_input()* immer fehlschlagen, ein negierter Vergleich dagegen immer zutreffen. Der Anwender kann die betreffenden Regeln für solche Fälle mit einem Schlüsselwort gezielt überspringen.

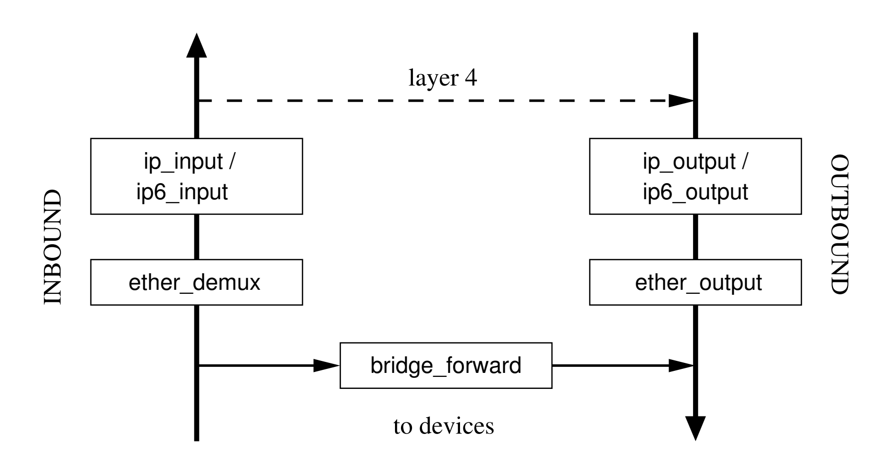


Abbildung 4.3: Paketflüsse durch FreeBSD IPFW

Bei IPFW in BSD/OS wurde nach [LLB02] die gesamte Paketverarbeitung neben den bereits bekannte Methoden *ip_input(), ip_output()* zusätzlich von *ip_forward()* durchgeführt. Die in Abbildung 4.4 dargestellten Zugriffspunkte entsprechen unter-

schiedlichen logischen Phasen innerhalb dieser Methoden, auf die eine Konfiguration oder der Benutzer nur indirekt Einfluss hat.

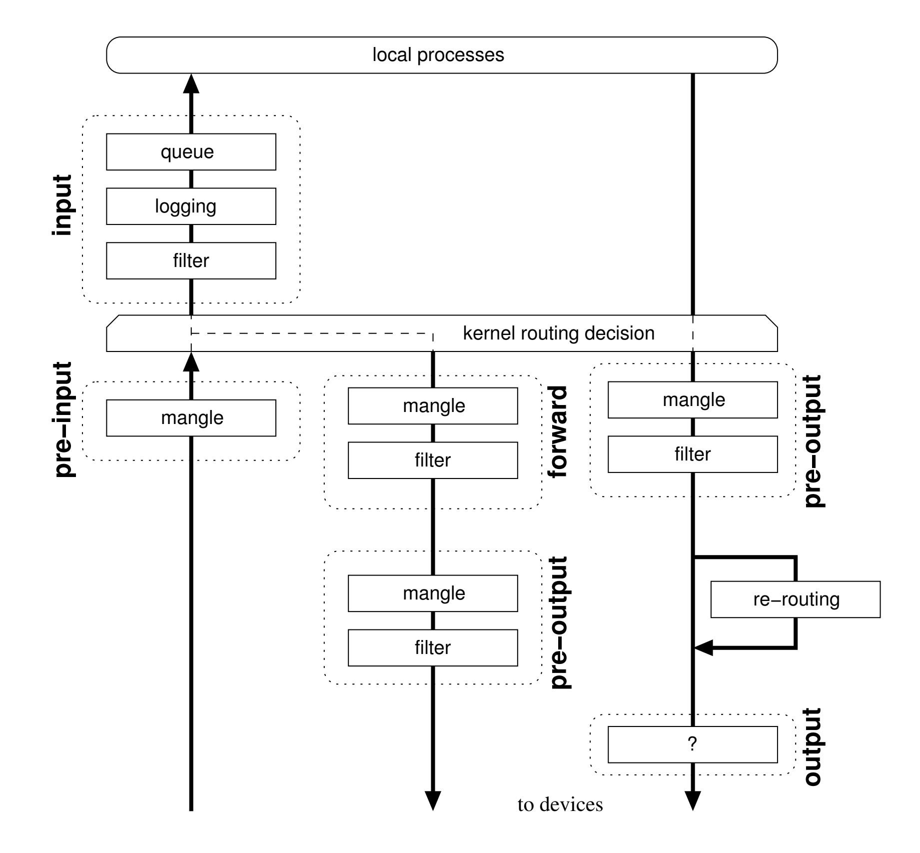


Abbildung 4.4: Paketflüsse durch BSD/OS IPFW

Alle Pakete wurden zunächst in der `pre-input`-Phase behandelt, wo sie verändert werden durften. Anschließend wurde beim Routing festgestellt, ob ein Paket für das lokale System bestimmt war oder weitergeleitet werden sollte. Die lokalen Pakete konnten in der `input`-Phase gefiltert, geloggt und an Benutzerprogramme weitergereicht werden. Pakete, die weitergeleitet wurden, konnten in den `forward` und `pre-output`-Phasen gefiltert oder erneut verändert bzw. umadressiert werden. Hier sollte aber die Umadressierung nicht mehr dazu führen, das Paket für das lokale System zu bestimmen. Für lokal erstellte Pakete wurde zunächst beim Routing eine ausgehende Netzwerkschnittstelle und der nächste Hop bestimmt, bevor sie im `pre-output` verändert und gefiltert werden konnten. Bei einer Umadressierung wurden die Pakete wieder an das Routing weitergereicht. Im Gegensatz zu weitergeleiteten Paketen wurden ausgehende Pakete zusätzlich in der `output`-Phase behandelt.

IPF - IPFilter

IPFilter läßt sich unter zahlreichen Betriebssystemen einbinden, weswegen eine eindeutige Positionierung der Paketverarbeitung im Kernel schwierig ist. Interessant zu bemerken ist, dass die IPFilter-Funktionen unter Linux/netfilter über die Zugriffs-

punkte PREROUTING und POSTROUTING auf die Pakete zugreifen. Die folgende Darstellung basiert vorwiegend auf der Projektdokumentation[2] sowie der Quellcodeanalyse.

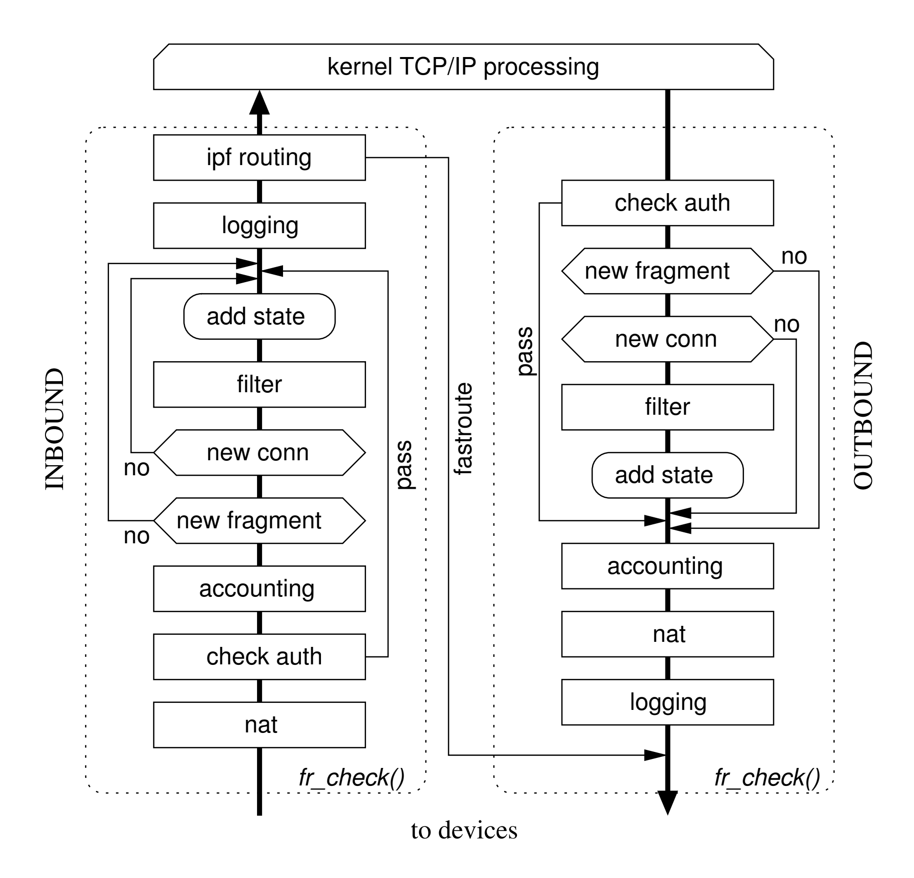


Abbildung 4.5: Paketflüsse durch IPFilter

Der Paketfluss bei IPFilter, dargestellt in Abbildung 4.5, betrachtet alle Pakete als ein- oder ausgehend, was sich auch jeweils in einer Regelliste für input und output wiederspiegelt. Weitergeleitete Pakete durchlaufen beide Bearbeitungsphasen. Dafür gibt es aber die Möglichkeit das Routing im IP-Stack über die Schlüsselwörter *to* oder das Synonym *fastroute*[3] zu umgehen, wobei IPF selbst das Paket routet und an den nächsten Hop adressiert.

Durch eine *auth*-Regel ist es möglich, ein externes Programm zur Benutzerauthentifizierung aufzurufen, welches dann die Filterentscheidung trifft. Ein so akzeptiertes Paket bzw. die dazugehörende Assoziation überspringt alle weiteren Überprüfungen in der bearbeiteten Richtung.

[2]Der beschriebene Paketfluss basiert auf der Dokumentation unter `http://coombs.anu.edu.au/~avalon/`.

[3] fastroute wird vor allem für einen *stealth*-Modus verwendet, bei dem die IP TTL nicht vermindert wird und dadurch die Firewall als zusätzliche Station unsichtbar bleibt.

PF - Packet Filter

Für die Analyse des Packet Filters wurden die Projektdokumentation und der Quellcode[4] herangezogen.

Eine Besonderheit in den Funktionen von PF und damit auch im Paketfluss, der in Abbildung 4.6 zu sehen ist, sind die eingebauten zusammengefassten Mechanismen zur Paketmodifikation. Einerseits können mit *scrub* die IP TTL und ID, die Maximum Segment Size (MSS) für TCP moduliert und andererseits die Fragmentflags und Fragmentierung bereinigt werden. Für TCP sind zudem Modulation der Sequenznummer und der Einsatz eines SYN-Proxies möglich, die mit der Zustandstabelle bzw. Zustandserstellung verknüpft sind. Diese können regelbasiert mit *keep state*, *modulate state* und *synproxy state* aktiviert werden.

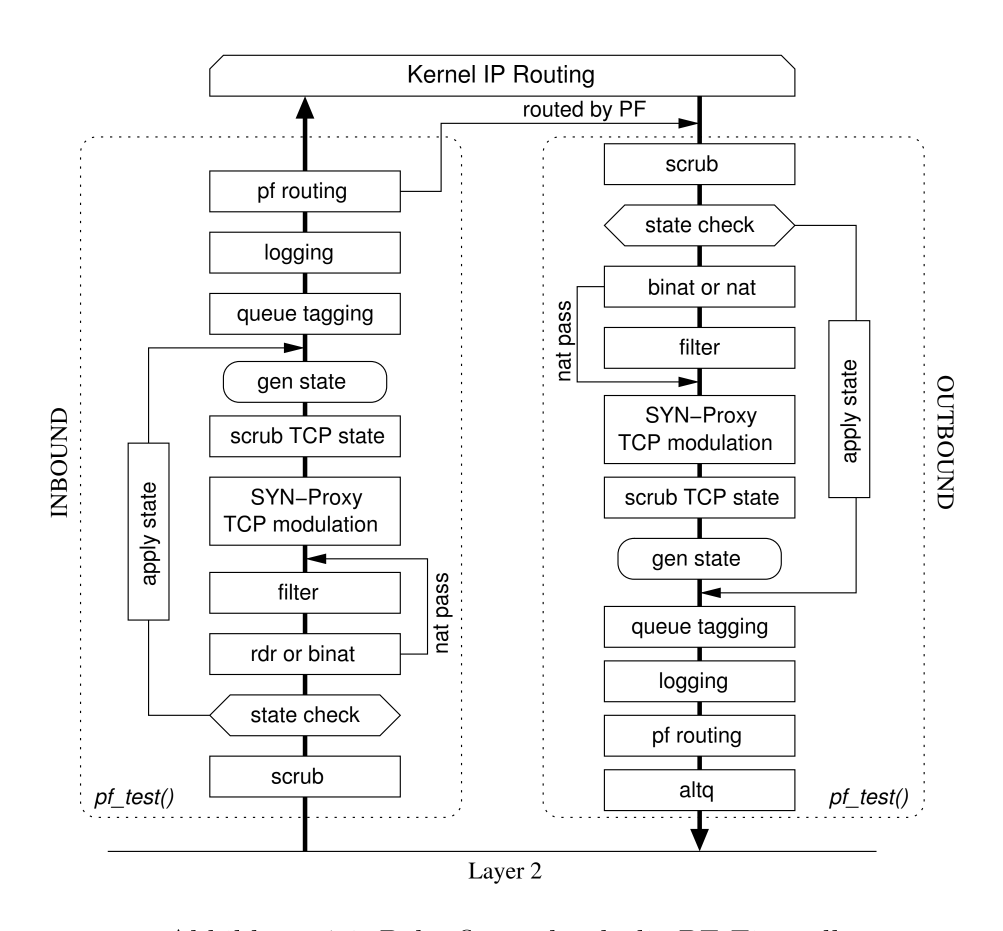


Abbildung 4.6: Paketflüsse durch die PF Firewall

Obwohl der OpenBSD Packet Filter syntaktische Ähnlichkeiten zum IPFilter aufweist, hat die *fastroute*-Option jeweils verschiedene Bedeutungen. Bei PF wird ein so markiertes Paket trotzdem nochmals in der Ausgangsrichtung verarbeitet. Bei IPFilter werden alle weiteren Schritte übersprungen und das Paket direkt weitergeleitet.

[4]Analysiert wurden `src/sys/net/pf_norm.c` (rev. 1.109) und `src/sys/net/pf.c` (rev. 1.550).

Cisco PIX

Die Analyse der Cisco PIX basiert auf dem offiziellen Handbuch und wurde mit Informationen aus [BCD04; Pru04] ergänzt.

Hauptmerkmale der PIX sind der Adaptive Security Algorithm (ASA), der die zustandbehaftete Paketfilterung und -veränderung durchführt und der Cut-Through Proxy, der die Benutzerauthentifizierung durchführt. Grundkonzept des ASA sind Sicherheitslevel. Sie werden vom Administrator den Netzwerkadaptern zugeordnet, wobei sie zur Umsetzung einer default Policy genutzt werden indem Netzwerkverkehr zunächst nur von einem höheren (inside) zu einem niedrigeren (outside) Level möglich ist. Alle weiteren Zugriffe müssen über access-lists und Adressumsetzung definiert werden. Dabei wird die Adressumsetzung als die Veröffentlichung von erreichbaren Adressen auf einer Netzwerkschnittstelle verstanden, so dass selbst bei gleichbleibenden Adressen eine Adressbekanntgabe in den Zonen mit niedrigerem Sicherheitslevel stattfinden muss.

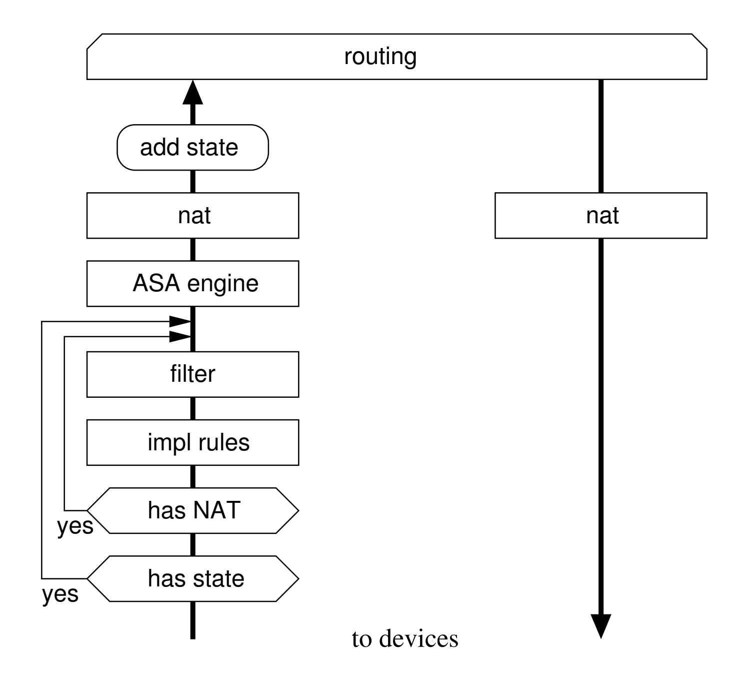


Abbildung 4.7: Paketflüsse durch die Cisco PIX

Abbildung 4.7 zeigt die Interpretation des Paketflusses nach den genannten Quellen. Die PIX unterscheidet zwei Zustandspeicher: einen für die Verbindungen (*conn table*) und einen für die durchgeführten NAPT-Umsetzungen (*xlate table*). Wird bei einem Vergleich des Paketes mit einer der Tabellen ein passender Eintrag gefunden, dann wird ein Vergleich mit den Filterregeln übersprungen. In jedem (weiterleitendem) Fall werden die Pakete von dem ASA überprüft, der den Anwendungsbereich mit statischen Mustern vergleicht und gegebenenfalls das Paket verwirft oder modifiziert. Nach diesen Überprüfungen wird DNAT angewendet, ein Zustand erstellt, das Routing und SNAT angewendet.

Checkpoint VPN-1/FireWall-1

Die Paketverarbeitung bei der FW-1 auf der SecurePlatform (Linux) wird in einem als Netzwerktreiber realisierten Hauptmodul, das die inneren Funktionen auf Betriebssystemebene zusammenfasst, durchgeführt. Dieses Hauptmodul greift je nach Konfiguration und Lizenzierung auf weitere Funktionsmodule zu, die sich nach [CPS03] wie in Abbildung 4.8 gezeigt, anordnen. Über die interne Verarbeitung konnten während dieser Untersuchung kaum Informationen gefunden werden. Zur weiteren Analyse wäre es aber möglich die eingebauten Monitoring- Funktionen, die jeweils zwischen den Funktionsmodulen eingeschaltet werden können, zu nutzen.

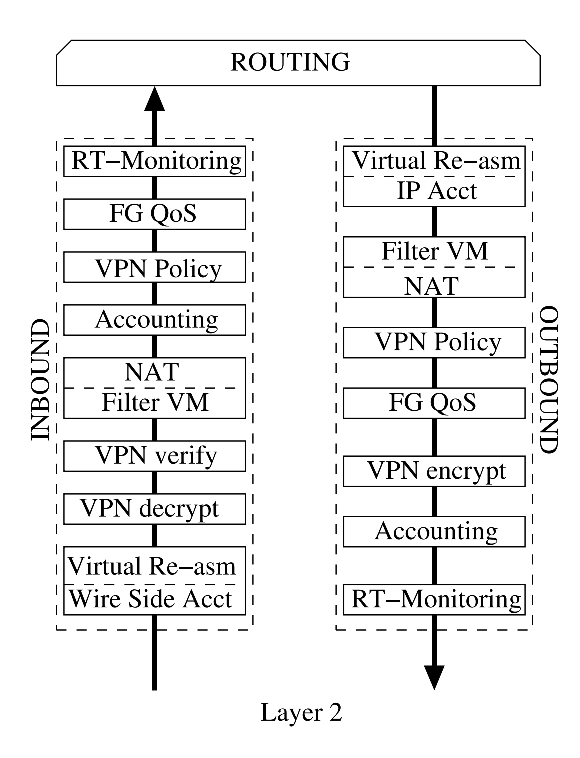


Abbildung 4.8: Paketflüsse durch FW-1

4.2. Zustandsbehaftete Verbindungsverfolgung

Im Abschnitt 2.1 wurden Firewalls als reglementierende Beobachter von Kommunikationsabläufen definiert. Sie sind vermittelnde Knoten zwischen den Endpunkten, die die Kommunikation beobachten können, sie bewerten und gegebenenfalls entsprechend ihrer Konfiguration eingreifen sollen. In diesem Abschnitt wird zuerst die grundlegende Notwendigkeit, die Protokollzustände der Endpunkte zu verfolgen, und die damit verbundene problematische Aufgabenstellung eingeführt. Danach werden die Umsetzungen der untersuchten Firewalls vorgestellt, die durch eine Analyse identifiziert wurden.

In ihrer beobachtenden Position zwischen den eigentlichen Endpunkten stehen die Firewalls in einer ungünstigen Position, in der sie nur eine ungefähre Vorstellung von den Protokollzuständen der Endpunkte haben. Üblicherweise werden Protokollabläufe für die miteinander kommunizierenden Endpunkte spezifiziert, wodurch sich nur indirekt und gegebenenfalls uneindeutig eine Protokollbeschreibung für den Zwischenpunkt ableiten lässt. Zwischen den beiden Kommunikationspartnern können mehrere solcher reglementierenden Instanzen positioniert sein, wobei jede von ihnen ihre eigene Vorstellung vom aktuellen Zustand der Assoziation und ihre eigene Richtlinie bzgl. der Reglementierung hat. Dadurch können die nachgelagerten Instanzen nur einen Ausschnitt aus der tatsächlichen Kommunikation beobachten, wodurch u.U. Uneindeutigkeiten entstehen.

Von außen gesehen kommunizieren zwei Teilnehmer miteinander, wobei die Firewall sich dazwischen befindet und versucht die Kommunikation zu beobachten sowie gegebenenfalls zu reglementieren. Dabei kann sie den korrekten Empfang von Nachrichten (und damit auch den zugeordneten Zustand des Endpunktes) nur aufgrund von entsprechenden Antworten der Gegenseite annehmen. Verlorene oder anderweitig geblockte Nachrichten kann sie nicht entdecken, was zu der Uneindeutigkeit führt. Abbildung 4.9 zeigt den TCP-Handshake und die Situation für Firewall B. Im Fall a) wäre ein erneutes SYN-Paket von A nach B, ein valides Vorgehen in der Protokollspezifikation. Im Fall b) hat A aber schon den Verbindungsaufbau abgeschlossen und ein weiteres SYN-Paket wäre invalid. Wäre dies ein Angriffsversuch sollte das Ablehnen eines solchen Paketes berücksichtigt werden. Hier kann dieser Zwischenpunkt aber die in den Endpunkten unterschiedlichen internen Zustände nicht unterscheiden und dementsprechend keine eindeutigen Entscheidungen treffen.

Ähnliches kann auch passieren, wenn Pakete asymmetrisch geroutet werden, wodurch der Zwischenpunkt nur einen Teil der Kommunikation sieht. Intern müssen trotzdem innerhalb der Firewall Entscheidungen getroffen werden, wie mit dem aktuellen und den folgenden Paketen umgegangen wird. Da die Firewall eine Schutzfunktion umsetzt, kann sie als diejenige Instanz betrachtet werden, die die Situation bewerten und reglementieren muss, also die alleinige Entscheidungskraft hat. Problematisch wird es, wenn mehrere solcher Knoten den gleichen Paketfluss reglementieren und sich u.U. gegenseitig, wie auch die Kommunikation behindern. Das kann dazu

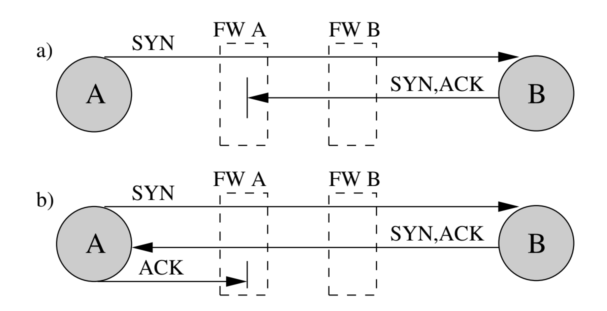


Abbildung 4.9: Uneindeutige Situationen im vermittelndem Knoten B

führen, dass auch die Protokollpartner nicht mehr korrekt kommunizieren können.[5]

Firewall-Hersteller haben aufgrund der fehlenden Protokollspezifikationen für Zwischenpunkte keine Richtlinien dafür, wie sie eine beobachtende Protokollinstanz implementieren sollten. Dabei unterliegen sie gleichzeitig einem Interessenkonflikt zwischen den Sicherheitsanforderungen an eine Firewall, die möglichst konservativ und streng reglementieren sollte, und der Benutzerfreundlichkeit bzw. Kompatibilität mit anderen Produkten, wodurch sie mehrere Interpretationen und Variationen von Protokollen zulassen müssen. Beide Aspekte führen zu unterschiedlichen Auslegungen, wie ein Zwischenpunkt ein bestimmtes Protokoll oder einen Paketfluss zu handhaben hat, wobei es nicht zwangsläufig ein richtig oder falsch gibt. In der Praxis wird eine Implementierung über Jahre hinweg den praktischen Erfahrungen und anderen Geräten angepasst, wodurch sich die gegenseitige Kompatibilität der Geräte erhöht. Trotzdem wird eine eindeutige Spezifikation benötigt, gegen die ein Zwischenpunkt geprüft werden kann. D. von Bidder-Senn beschäftigt sich mit diesem Problem und zeigt in [BS06] wie es möglich ist, Zwischenpunktspezifikationen aus den Endpunktspezifikationen systematisch zu erzeugen. Sie schlägt einen Algorithmus vor, der unter Eingabe von Automaten für die Endpunkte eines Protokolls einen Protokollautomaten für den Zwischenpunkt erzeugt. Mit so einem Automaten wäre es möglich eine Firewall zu konstruiren. Die vorliegende Arbeit jedoch nimmt die Sicht der Endpunkte ein und testet die Reaktion der Firewall auf die Kommunikation zwischen den beiden Protokollpartnern.

Alle Mechanismen von zustandsbehafteten Paketfiltern basieren auf der Verfolgung von Assoziationen zwischen den Endpunkten[6]. Dabei werden innerhalb einer Assoziation mehrere Phasen unterschieden, die bedingt durch die jeweilige Protokollspezifikation voneinander abhängen und deshalb von der Firewall durchgesetzt werden. Dadurch ist es möglich die Initiierung einer Assoziation nur von ausgewählten Endpunkten zu erlauben und die Rückrichtung nur für korrekte Antwortpakete zu öffnen, was eine erhöhte Kontrolle und striktere Filterung der Pakete erlaubt. Ebenso ist es dann möglich Kontroll- und Fehlernachrichten nur im Kontext einer bekannten Assoziation zuzulassen.

Damit dieser Mechanismus funktioniert müssen über die zugelassenen Assozia-

[5] Ein Beispiel für eine behindernde Reglementierung ist die Filterung von ICMP-Fehlernachrichten.
[6] Vgl. auch die Einführung der Firewalltypen auf Seite 7.

tionen Informationen, die diese eindeutig beschreiben, gesammelt werden. Im Regelfall sind das bei TCP- und UDP-Assoziationen die Richtung, die ein- und/oder ausgehende Schnittstelle, Quelladresse und -port, Zieladresse und -port sowie eine Repräsentation für die Phase der Assoziation und eine Gültigkeitsdauer. Bei ICMP-Nachrichten werden Typ und Code statt der Ports hinterlegt. Ein mögliches Beispiel für diese Zustandsinformationen ist in Listing 4.1 zu sehen.

```
out tcp 6 timeout=431976 ESTABLISHED
   src=192.168.1.1 dst=10.1.2.3 sport=50754 dport=5222
   pkts=1 bytes=313
out tcp 6 timeout=431981 ESTABLISHED ASSURED
   src=192.168.1.1 dst=192.168.1.2 sport=42666 dport=1863
   pkts=2 bytes=109
in udp 17 timeout=22 UNREPLIED
   src=192.168.1.2 dst=255.255.255.255 sport=138 dport=138
   pkts=1 bytes=242
```

Listing 4.1: Beispiel für Zustandsinformationen

Optional können noch weitere Informationen abgelegt werden. Dazu zählen: die erwarteten Antwortpakete, bei TCP die Angaben zu Sequenznummern und Flags, die traversierten Netzwerkschnittstellen, Informationen zu durchgeführten Adresstransformationen oder relevante Daten aus den Anwendungsschichten.

Bei der weiteren Verbindungsverfolgung sollten neben den IP-Adressen und den Ports auch die Sequenznummern berücksichtigt werden, um eine Fremdmanipulation einer Verbindung (z.B. durch Session-Hijacking) zu verhindern.

Die meisten dieser Informationen ergeben sich aus der Analyse des Pakets und des Paketflusses. Die Timeouts werden teilweise von den jeweiligen Protokollspezifikationen vorgegeben, sind aber auch Teil der Firewallkonfiguration. Sie können besonders großzügig oder besonders strikt vorgegeben sein. Je nach Freiheitsgrad der Konfigurationssprache können sie global, für eine Regel, für einen Dienst, ein Protokoll oder einen Protokollzustand eingestellt werden.

netfilter/iptables

netfilter/iptables hat zwei Möglichkeiten Verbindungszustandsinformationen zu verwalten. Das Modul *state* bietet eine vereinfachte Sicht und das Modul *conntrack* eine ausführliche Sicht auf die Assoziationszustände. Im ersten Fall stehen dem Benutzer die Zustände `INVALID, NEW, ESTABLISHED, RELATED` und `UNTRACKED` zur Verfügung, die wie folgt definiert sind:

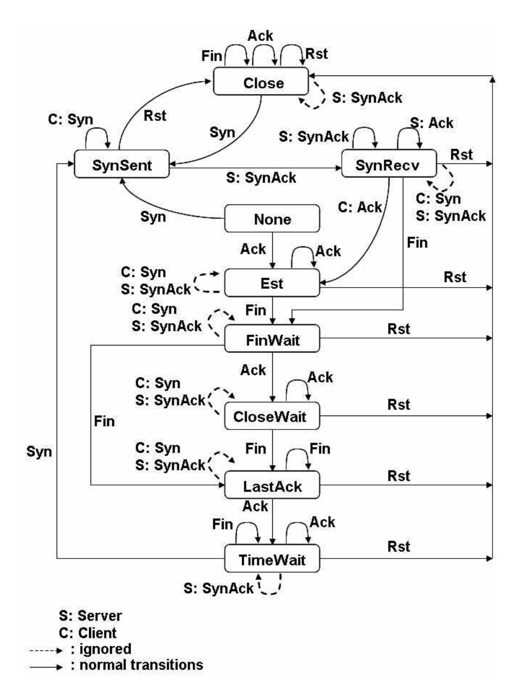
NEW Die Assoziation startet und der Zustand ist erreicht, wenn das Paket ein gültiges Initiierungspaket in der Sequenz zur Herstellung einer Assoziation ist und die Firewall einen Datenstrom nur in eine Richtung beobachtet hat.

ESTABLISHED Die Assoziation wurde hergestellt und die Firewall hat Datenstrom in beiden Richtungen gesehen.

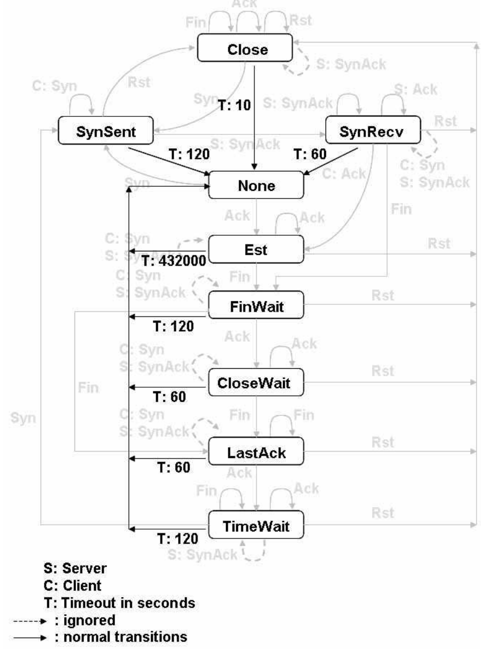
RELATED Das ist eine erwartete Assoziation im Kontext einer bereits hergestellten Assoziation. Darunter werden zum Beispiel ICMP-Fehlernachrichten oder ein FTP-Datenkanal im Kontext einer FTP-Kontrollverbindung eingestuft.

INVALID Das ist ein Spezialzustand, der für unerwartete Pakete benutzt wird. Das könnten z.B. ICMP-Fehlernachrichten ohne Kontext oder fehlerhafte Pakete sein.

UNTRACKED Dieser Zustand ist mit der *raw*-Tabelle hinzugefügt worden, um Pakete zu markieren, die durch entsprechende Regeln von der Verbindungsverfolgung ausgeschlossen wurden. Dies ermöglicht Ressourcen zu sparen, falls für bestimmte Assoziationen keine detaillierten Sicherheitsanforderungen bestehen.



(a) Vereinfachte Zustandsmaschine mit Übergängen durch Senden von TCP Flags



(b) Vollständige Zustandsmaschine mit Timeout-gesteuerten Übergängen

Abbildung 4.10: Darstellung der iptables TCP-Zustandsmaschine

Das mächtigere Modul *conntrack* ermöglicht es zusätzlich Pakete, bei denen SNAT oder DNAT angewendet wurde sowie die detaillierten internen Zustände (`EXPECTED`, `UNREPLIED`, `ASSURED`, `NONE`, ...) der jeweiligen Protokoll-Zustandsmaschine zur Verbindungsverfolgung abzufragen. Für die komplexere Zustandsmaschine von TCP sind die gültigen Übergänge in Abbildung 4.10 dargestellt[7]. Bei den Transitionen werden drei Typen unterschieden: ein Zustandsübergang aufgrund einer validen Eingabe,

[7] Die Abbildung basiert auf der Analyse des Quellcodes von `linux/net/ipv4/netfilter/ip_conntrack_proto_tcp.c` Version 2.2 in Linux 2.6.20.

kein Zustandsübergang für ignorierte Eingaben und kein Zustandsübergang bei invaliden Eingaben. Der Unterschied zwischen validen und ignorierten Eingaben ist, dass erstere eine Erneuerung des mit dem Zustand verknüpften Timeouts bewirken, wogegen die ignorierten Eingaben die Eingabe akzeptieren, aber den Timeout nicht verändern.

Die TCP-Flags PUSH und URGENT werden bei den Übergängen zwischen den Zuständen nur indirekt bei iptables berücksichtigt, da alle gesetzten Flags mit einer Tabelle von validen Kombinationen verglichen wird, bei der auch PUSH und URGENT eingetragen sind.

Eine solche Zustandsmaschine konnte nur bei netfilter identifiziert werden. Bei den anderen drei im Quellcode verfügbaren Paketfiltern werden an verschiedenen Stellen einzelne TCP-Flags verglichen. Über die kommerziellen Paketfilter kann keine Aussage gemacht werden.

Die Verbindungsverfolgung für UDP unterscheidet aufgrund der verbindungslosen Natur des Protokolls nur zwischen den einfacheren Zuständen `NEW`, `ESTABLISHED` und `RELATED` Assoziationen – letztere beiden werden auch nach Senderichtung unterschieden. Bei ICMP werden den drei Zuständen bestimmte ICMP-Typen zugeordnet. Vier Frage-Antwort-Paare werden als `NEW` bzw. `ESTABLISHED` und fünf Fehlerbenachrichtigungen für bestehende Assoziationen als `RELATED` klassifiziert. Bei den Paaren wird intern ein Zähler dazu verwendet, den Eintrag nur so lange zu erhalten, wie Antworten auf gesendete Anfragen ausstehen.

```
tcp 6 431992 ESTABLISHED
   src=10.0.1.1 dst=10.0.2.2 sport=9575 dport=1863 pkts=2 bytes=109
   src=10.0.2.2 dst=10.0.1.1 sport=1863 dport=9575 pkts=1 bytes=60 [ASSURED]
   mark=0 use=1
tcp 6 93 SYN_SENT
   src=10.0.1.1 dst=10.0.2.2 sport=9546 dport=631 pkts=1 bytes=60 [UNREPLIED]
   src=10.0.2.2 dst=10.0.1.1 sport=631 dport=9546 pkts=0 bytes=0
   mark=0 use=1
udp 17 178
   src=10.0.1.1 dst=10.0.2.2 sport=2869 dport=53 pkts=2 bytes=130
   src=10.0.2.2 dst=10.0.1.1 sport=53 dport=2869 pkts=2 bytes=234 [ASSURED]
   mark=0 use=1
icmp 1 30
   src=10.0.2.2 dst=10.0.2.255 type=8 code=0 id=352 pkts=1 bytes=84 [UNREPLIED]
   src=10.0.2.255 dst=10.0.2.2 type=0 code=0 id=352 pkts=0 bytes=0
   mark=0 use=1
```

Listing 4.2: Beispiel für iptables Zustandsinformationen (umformatiert)

Das Kommandozeilenprogramm iptables wird nur zur Konfiguration des Regelwerks selbst verwendet. Statusinformationen, Parameter und Schalter zur Steuerung des Verhaltens von netfilter und des Netzwerkstapels werden dagegen im virtuellen *proc*-Dateisystem abgebildet. Bei Linux ist es standardmäßig unter `/proc` eingebunden. Unterhalb von `/proc/net` werden nur lesbare Statusinformationen hinterlegt

und in `/proc/sys/net` befinden sich auch modifizierbare Parameter, die mit Standardwerten vorbelegt sind. Besonders interessant ist hier die *conntrack*-Tabelle, die die gerade aktiven Assoziationen beinhaltet und wie eine Datei ausgelesen werden kann. Eine beispielhafte Ausgabe ist in Listing 4.2 zu sehen.

IPFW - IP-FireWall

Die zustandbehaftete Verbindungsverfolgung bei IPFW wird über dynamische und zeitlich begrenzte Regeln, die sich auf die im Paket beschriebenen Endpunkte beziehen, beim Filtern realisiert. Zustandsinformationen werden bei jeder akzeptierenden Regel mit einem der Schlüsselwörter *keep-state* oder *limit* erstellt und beim ersten Auftreten von *keep-state* oder *check-state* überprüft. Somit kann der Zeitpunkt der Überprüfung eingeschränkt beeinflusst werden. In Listing 4.3 ist ein Ausschnitt aus einem Regelwerk incl. aktiver dynamischer Regeln darsgestellt, der mit dem Befehl `ipfw -ad show` erzeugt wurde.

Die abgelegte Informationsmenge und Wirkung ist mit dem bereits vorgestellten *conntrack*-Mechanismus bei netfilter/iptables vergleichbar. Allerdings hat bei IPFW der Benutzer keinen Zugriff auf die internen Zustände bei der Verbindungverfolgung und kann sie auch nicht explizit in dem Regelwerk ansprechen.

```
## nr   in    out rule
00100    0      0 check-state
00200  674  57241 allow tcp from 10.0.1.0/24 to any keep-state
00300   46   3674 allow udp from 10.0.1.0/24 to any keep-state
00400   34   2856 allow icmp from 10.0.1.0/24 to any keep-state
65535   42   4754 deny ip from any to any
## Dynamic rules (24):
00400    5    420 (5s) STATE icmp 10.0.1.11 0 <-> 10.0.1.1 0
00200  673  57181 (300s) STATE tcp 10.0.1.11 34022 <-> 10.0.1.1 22
```

Listing 4.3: Beispiel für IPFW Zustandsinformationen

Über die internen Mechanismen zur Erstellung der dynamischen Regeln und der tatsächlich berücksichtigten Paketeigenschaften ist keine Dokumentation vorhanden.

IPFilter

IPFilter kann bei jeder Regel angewiesen werden Zustands- und Fragmentinformationen (*keep state* und *keep frags*) zu sammeln, die anschließend automatisch verwendet werden und dazu führen, dass folgende Pakete mit den gespeicherten Einträgen verglichen werden und eine erneute Regelauswertung übersprungen wird.

Beim Accounting und der Zustandsgenerierung werden sehr viele Informationen gesammelt, die geloggt und abgerufen werden können. Für das Logging wird `ipmon` verwendet, das separat aktuelle Änderungen der Zustands- und NAT-tabelle sowie die Filterentscheidungen ausgeben kann. Eine Beispielausgabe ist in Listing 4.4 zu se-

hen. Detaillierte Zustandsinformationen können zusätzlich mit dem Befehl ipfstat –sl angezeigt werden und sind in Tabelle 4.1 veranschaulicht.

IPFilter war einer der führenden Paketfilter, der erweiterte Eigenschaften des TCP Protokolls in die zustandsbehaftete Verbindungsverfolgung eingebracht hat. Dazu gehört das Verfolgen der gültigen und noch unbestätigten Sequenznummern innerhalb des aktiven TCP-Fensters. Die Basis für diesen Mechanismus wurde in der Untersuchung von Guido van Rooij [Roo00] beschrieben.

```
 # ipmon -o S
[tstamp] STATE:NEW     10.1.1.1,53 -> 10.2.2.15,53 PR udp
[tstamp] STATE:EXPIRE  10.1.1.1,53 -> 10.2.2.15,53 PR udp
  Pkts 4 Bytes 356
[tstamp] STATE:NEW     10.2.2.13,1119 -> 10.1.1.10,22 PR tcp
[tstamp] STATE:EXPIRE  10.2.2.13,1119 -> 10.1.1.10,22 PR tcp
  Pkts 63 Bytes 4604

 # ipmon -o N
[tstamp] @2 NAT:RDR     10.2.2.25,113 <--> 10.2.2.25,113 [10.1.1.13,45816]
[tstamp] @2 NAT:EXPIRE  10.2.2.25,113 <--> 10.2.2.25,113 [10.1.1.13,45816]
  Pkts 10 Bytes 455

 # ipmon -o I
[tstamp] ppp0 @0:2 b 10.1.1.103,443 -> 20.2.2.10,4923
  PR tcp len 20 1488 -A
[tstamp] xl0 @0:1 S 10.2.2.254 -> 255.255.255.255
  PR icmp len 20 9216 icmp 9/0
```

Listing 4.4: Beispiele für IPF Zustandsinformationen

Art	gesammelte Informationen
Adressierung	Quell-IP und -port, Ziel-IP und -port, Schnittstellen
Zustand	aktuell/gesamt, Speicherbehälternummer
Protokoll	IP-Version, Protokollnummer
IP	IP-Optionen, TTL
TCP	aktuelle und Start-Sequenznummern, TCP-Window, Window-Scale, ISN-Offsets, Flags
ICMP	Typ und Code
Filter	pass/block, Regeloptionen
Statistik	Paket- und Bytezähler in/out
Flags	Sync-Status, Markierungen

Tabelle 4.1: Zusammenfassung der Zustandsinformationen bei ipfstat

PF - Packet Filter

Die Dokumentation zum Kontrollprogramm *pfctl* zeigt viele Beispiele zur Anzeige und Auswertung der umfangreich gesammelten Zustandsdaten. Neben den aktiven

NAT-Regeln, den Einträgen in der Zustands- und der source tracking-Tabelle werden auch viele Accounting-Daten gesammelt. Diese können nach NAT, nach geladenen Queues, nach Filterregel, nach Tabelle, nach Markierung und nach Netzwerkschnittstelle aufgeschlüsselt werden. Ein Ausschnitt der Zustandstabelle ist in Listing 4.5 zu sehen, und die möglichen Zustände sind in Tabelle 4.2 aufgeschlüsselt.

```
self icmp 10.0.1.2:2755 <- 10.0.1.1                          0:0
self tcp  127.0.0.1:113 <- 127.0.0.1:54358         TIME_WAIT:TIME_WAIT
self tcp  127.0.0.1:54358 <- 127.0.0.1:113            CLOSED:SYN_SENT
self tcp  10.0.1.2:22 <- 10.0.1.4:33905                  EST:EST
self udp  255.255.255.255:138 <- 10.0.1.2:138     NO_TRAFFIC:SINGLE
self udp  10.0.1.2:10318 <- 10.0.1.21:7594          MULTIPLE:MULTIPLE
self udp  10.0.1.2:23449 <- 10.0.1.21:7594            SINGLE:MULTIPLE
all  tcp  10.0.1.21:2212 -> 10.0.1.1:55070 -> 10.5.5.5:22 EST:EST
```

Listing 4.5: Beispiel für PF Zustandsinformationen

Zustand	t in sec	Beschreibung
tcp.first	120	nach dem ersten Paket
tcp.opening	30	bevor das Ziel nur ein Paket gesendet hat
tcp.established	86400	vollständig aufgebaute Verbindung
tcp.closing	900	nach dem ersten FIN Paket
tcp.finwait	45	nachdem beide Seiten FIN gesendet haben. Die Verbindung ist geschlossen
tcp.closed	90	nachdem ein RST gesichtet wurde
udp.first	60	nach dem ersten Paket
udp.single	30	nachdem nur eine Seite mehrere Pakete verschickt hat
udp.multiple	60	nachdem beide Seiten Pakete geschickt haben
icmp.first	20	nach dem ersten Paket
icmp.error	10	nachdem eine ICMP-Fehlernachricht als Antwort auf ein ICMP Paket gesichtet wurde
other.first	60	nach dem ersten Paket
other.single	30	nachdem nur eine Seite mehrere Pakete verschickt hat
other.multiple	60	nachdem beide Seiten Pakete geschickt haben

Tabelle 4.2: Aufschlüsselung der Protokollzustände bei PF

Das Verhalten der Verbindungsverfolgung lässt sich sehr umfangreich anpassen. Neben den festen und adaptiven Gültigkeitsdauern für Einträge sind auch Beschränkungen der maximalen Anzahl von Zuständen, von Fragmenten und von Einträgen pro Quelladresse regulierbar – jeweils global und pro Regel. Daneben kann für Zustandseinträge eine Alterungsrichtlinie aus *normal, high-latency, aggressive* und *conservative* sowie die Bindung an eine konkrete Netzwerkschnittstelle, eine Gruppe (z.B. alle PPP-Schnittstellen) oder keine Bindung gewählt werden.

Cisco PIX

Alle Informationen zu durchgeführten Adress- und Portumsetzungen werden vom ASA in der xlate Tabelle gespeichert. Die Verbindungsinformationen werden in einer Zustandstabelle abgelegt. Die Zustandsverfolgung vom ASA ist verbindungsorientiert, weshalb bei TCP auch die Sequenznummern abgelegt und verglichen werden, was die externe Manipulation von Zustandsdaten erschwert. Die initialen Sequenznummern werden standardmäßig vom ASA randomisiert.

```
pixfirewall(config)# show conn detail
2 in use, 2 most used
Flags:
 A - awaiting inside ACK to SYN, a - awaiting outside ACK to SYN,
 B - initial SYN from outside,    C - CTIBQE media,
 D - DNS,                         d - dump,
 E - outside back connection,     f - inside FIN,
 F - outside FIN,                 G - group,
 g - MGCP,                        H - H.323,
 h - H.255.0,                     I - inbound data,
 i - incomplete,                  k - Skinny media,
 M - SMTP data,                   m - SIP media
 O - outbound data,               P - inside back connection,
 q - SQL*Net data,                R - outside acknowledged FIN,
 R - UDP RPC,                     r - inside acknowledged FIN,
 S - awaiting inside SYN,         s - awaiting outside SYN,
 T - SIP,                         t - SIP transient,
 U - up
TCP outside:192.150.49.10/23 inside:10.1.1.15/1026 flags UIO
UDP outside:192.150.49.10/31649 inside:10.1.1.15/1028 flags dD
```

Listing 4.6: Beispiel für Cisco PIX Zustandsinformationen

Listing 4.6 zeigt die Ausgabe der Zustands- und Verbindungsinformationen, die über alphabetische Flags gekennzeichnet werden. Demnach würde ein TCP-Handshake von *inside* nach *outside* nach dem SYN als 'saA', nach dem SYN-ACK als 'A', nach dem ACK als 'U' gekennzeichnet werden. Nachdem erfolgreich Daten ausgetauscht worden wären, würden noch 'I' bzw. 'O' hinzugefügt werden. Leider ist aus der Dokumentation nicht ersichtlich, wie weit die interne Darstellung von der hier gezeigten abweicht und es konnten keine weiteren Daten ermittelt werden.

Während der Untersuchung wurde beobachtet, dass die Cisco PIX für über VPN-Verbindungen authentifizierte Endpunkte dynamisch Regeln erstellt, die in der ausführlichen Regelausgabe in der Form

```
access-list dynacl<num> permit ip any host <vpnhost>
```

sichtbar sind.

Checkpoint VPN-1/FireWall-1

Die Zustandsinformationen werden bei der FW-1 im Hexadezimalformat dargestellt, wodurch sie für den Benutzer schwer zu interpretieren sind. Zudem müssen manche Felder als Ganzes und andere als zusammengesetztes Feld interpretiert werden. Die Beschreibung des Formats basiert im Folgenden auf [CPS03] und kann im Bedarfsfall dort detailliert nachgelesen werden. Die Ausgabe erfolgt über fw tab –s state.

Die ersten sechs Felder bis zum Semikolon bilden einen Schlüssel, über den die folgenden Werte referenziert werden können. Sie beschreiben die initiale Verbindungsrichtung, die Quelladresse und den Quellport, die Zieladresse und den Zielport sowie die Nummer des IP Protokolls. Die weiteren Felder werden in Tabelle 4.3 benannt.

Feld	Beschreibung
1–6	eindeutiger Schlüssel
7	type/r_ctype (zusammengesetztes Feld)
8	flags/r_cflags (hex)
9	Regelnummer, die zu dem Eintrag führte
10	vorgegebener timeout für die Assoziation
11	Adresse der Behandlungsroutine für Pakete dieser Assoziation
12–15	eindeutige ID der Assoziation
16	ID der eingehenden Schnittstelle auf Client-Seite
17	ID der ausgehenden Schnittstelle auf Client-Seite
18	ID der eingehenden Schnittstelle auf Server-Seite
19	ID der ausgehenden Schnittstelle auf Server-Seite
20–n_{-1}	IDs der Kernelbuffer
n	Zeit: verbliebene Gültigkeit/gesamte Gültigkeit

Tabelle 4.3: Feldbeschreibungen in Zustandsinformationen bei FW-1

Seit Version NG von VPN-1/FireWall-1 werden in der Zustandstabelle auch symbolische Verknüpfungen angelegt. Diese benutzen die gleiche Art von Schlüssel wie reguläre Einträge und verweisen auf einen ebensolchen Schlüssel. Zusätzlich haben sie noch einen Wert, der den Typen der Verknüpfung angibt. Die meisten Assoziationen werden durch vier Einträge in der Zustandstabelle abgebildet, die die ein- und ausgehenden Assoziationsdaten auf Clientseite und die Rückrichtung beschreiben. Nur die eingehende Assoziation vom Client erzeugt einen regulären Eintrag und die anderen verweisen jeweils darauf.

```
<00000001, d4968d33, 000003fc, d496c1dc, 00000801, 00000011; 00020001,
 00020001, 06000000, 00000028, 00000000, 3bb7aea0, 00000001, d4968d33,
 000007b6, ffffffff, ffffffff, 00000001, 00000001, 00000000, 00000000,
 00000000, 00000000, 00000000, 00000000, 00000000, 00000000; 27/40>
<00000000, d496c1dc, 00000801, d4968d33, 000003fc, 00000011>
 -> <00000001, d4968d33, 000003fc, d496c1dc, 00000801, 00000011>
    (00000006)
```

Listing 4.7: Beispiel für Checkpoint VPN-1/FW-1 Zustandsinformationen

5. Allgemeines Modell für zustandsbehaftete Paketfilter

In diesem Kapitel wird ein allgemeines Modell für zustandsbehaftete Paketfilter entwickelt. Unter allgemein wird dabei ein Modell verstanden, das nicht nur spezifische oder gezielt ausgewählte Objekte abbilden kann. Deswegen wurden in Abschnitt 2.3 mehrere Firewallsysteme ausgewählt, die für die in der Praxis typisch eingesetzten Systeme repräsentativ sind. Weiterhin wurden in Kapitel 3 die Merkmale der ausgewählten Systeme betrachtet und in Abschnitt 3.3 eine Bewertung sowie eine erste Klassifizierung der angebotenen Funktionen getroffen. Dadurch konnten die Merkmale in typische und weniger verbreitete Funktionen bzw. Auswahlkriterien unterteilt werden.

In Kapitel 4 wurde die Funktionsweise der Paketfilter untersucht. Dabei wurden in Abschnitt 4.1 anhand der verfügbaren Dokumentation die Paketflüsse durch die Firewalls identifiziert und in Abschnitt 4.2 die Mechanismen der Verbindungsverfolgung analysiert. In diesem Kapitel wird jetzt ein Schritt von der Analyse zur Entwicklung eines Baukastenmodells gemacht, mit dem sich die Paketflüsse und die Funktionen der zustandsbehafteten Paketfilter allgemein und vereinheitlicht beschreiben lassen. Diese Verallgemeinerung ergibt sich aus den Aufgaben von vermittelnden Paketfiltern[1], weswegen sie alle im Kern ein vergleichbares Vorgehen aufweisen.

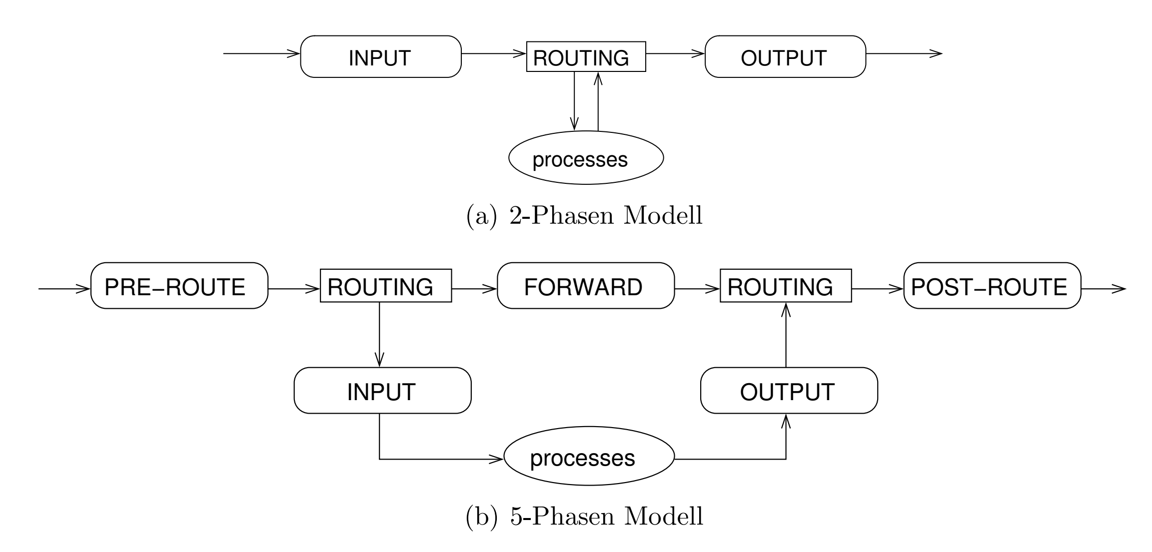
Abschnitt 5.1 stellt die entwickelten Bausteine des Modells auf verschiedenen Abstraktionsebenen vor. In Abschnitt 5.2 werden diese Bausteine verwendet, ausgewählte Funktionen zu modellieren. Abschnitt 5.3 zeigt beispielhaft die vollständige Modellierung des netfilter/iptables Paketfilters.

5.1. Bausteine des Modells

In Abschnitt 4.2 wurden mehrere Arbeitsabläufe bzw. Paketflüsse vorgestellt, die auf zwei immer wiederkehrende Ablaufdiagramme bei der Paketbearbeitung abgebildet werden können. Der dominierende Ablauf, der bei den Firewalls PF, IP-Filter, FreeBSD IPFireWall, PIX und FW1 beobachtet werden kann, besteht aus den drei Phasen `Input`, `Routing` und `Output`. Netfilter/iptables und die BSD/OS-Variante von IPFW dagegen arbeiten mit fünf Phasen, die als `Preprocessing`, `Input`, `Forward`, `Output` und `Postprocessing` vereinheitlicht benannt werden können. Die Anzahl der Phasen bestimmt, wie oft Pakete vom Netzwerkstapel bzw. Kernel an den Paketfilter übergeben werden. Dementsprechend stellen die einzelnen Phasen

[1] Vgl. hierzu die Einführung der Firewalltypen in Abschnitt 2.1.

Übergabepunkte zwischen den verarbeitenden Instanzen bzw. Zugriffspunkte des Paketfilters auf die Pakete und die damit verbundenen Datenstrukturen dar.



(a) 2-Phasen Modell

(b) 5-Phasen Modell

Abbildung 5.1: Zwei Modelle der Paketverarbeitungsphasen

In jeder der Phasen werden verschiedene Arbeitsschritte der Paketverarbeitung ausgeführt. Wie aus der Zusammenfassung der Analyse in Abschnitt 3.3 hervorgeht, sind diese Funktionen und die Mächtigkeit der einzelnen Firewallsysteme vergleichbar. Im Groben lassen sich diese Funktionen in die Funktionsgruppen Filterentscheidungen, (Re-)Routing, Paketmodifikationen, Adressumsetzung, Markierungen, Logging und Accounting sowie Weiterbehandlung durch weitere Prozesse unterteilen (vgl. auch Tabelle 3.3). Die Arbeitsschritte sind je nach Paketfilter unterschiedlich ausgeprägt und konfigurierbar. Sie sind eine Verfeinerung der Kernfunktionen eines Paketfilters wie sie bereits in Abbildung 4.1 definiert wurden.

Eine spezielle Art der Anweisungen dient der Strukturierung des Regelwerks, die einen Entscheidungsgraphen mit Verzweigungen aufbauen. Auch die Filterentscheidungen können den weiteren Verlauf im Entscheidungsgraphen und damit auch im Paketfluss beeinflussen. Deswegen wird für die Modellierung der Arbeitsschritte zunächst die Modellierung der Regelwerk-verändernden Funktionen benötigt.

Die Regeln eines Regelwerks bilden Knoten in einem Entscheidungsgraphen. Jede Regel besteht aus einem Bedingungsteil und einer Anweisung, die den weiteren Pfad beeinflusst. Falls der Bedingungsteil zutrifft, kann eine Anweisung den Graphen verzweigen, zusammenführen, fortführen oder abbrechen. Anweisungen können gruppiert werden, um Verzweigungs- bzw. Anspringpunkte zu realisieren. Das wird firewall-spezifisch durch die Bearbeitungsphasen vorgegeben und kann durch benutzerdefinierte Regelgruppen (vgl. Abschnitt 3.2) verfeinert werden. Im Folgenden werden für Arbeitsschritte der Begriff *task* und für Regelgruppen die von iptables übernommene Bezeichnung *chain* verwendet.

In Abbildung 5.2 wird eine symbolische Repräsentation der Regeln eingeführt und dazu verwendet zwei unterschiedliche Arten der Verzweigung im Regelwerk vorzu-

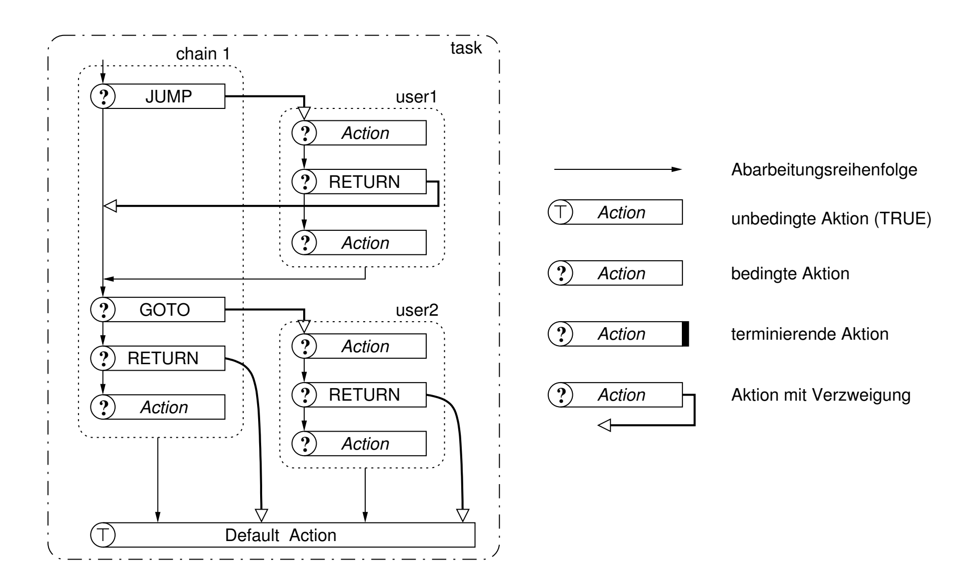


Abbildung 5.2: Verzweigungsarten bei der Regelauswertung

stellen. Diese zwei Arten sind ebenfalls an iptables angelehnt. Das iptables JUMP ist von der Wirkung her ähnlich zum `jump subroutine` (jsr) aus der Assembler-Programmierung. Dabei wird die Position in der aufrufenden chain gespeichert und bei explizitem RETURN bzw. Ende der aufgerufenen chain wieder fortgesetzt. Der Sprungbefehl GOTO dagegen ist mit dem einfachen `jump` (jmp) vergleichbar, rettet nicht die Position der gerade aufrufenden chain und setzt beim Rücksprung die Bearbeitung bei der zuletzt geretteten Position fort. Bei iptables wird die Standard-Richtlinie (default policy) als oberstes Rücksprungziel vorgegeben. Die Verzweigungen anderer Firewalls lassen sich auf diese Verzweigungsarten abbilden.

Das Ergebnis einer Anweisung kann, wie bereits erwähnt, den Entscheidungspfad beeinflussen, indem die Bearbeitung nach einer Entscheidung für das betrachtete Ereignis (Paket) fortgeführt oder abgebrochen wird. Damit wird neben einer funktionellen Klassifizierung der Anweisungen eine weitere eingeführt, die den Paketfluss berücksichtigt.

Art	Beispiele
terminierend	Drop und Reject
nicht-terminierend	Modifikationen von IP-Optionen, TTL, Fragmentierungsbits, TOS-Feldern sowie TCP Sequenznummern und MSS; interne Markierungen, Logging und Accounting
weiterleitend	Accept, verschiedene NAT-Arten, (Re-)Routing

Tabelle 5.1: Entscheidungsarten eines Paketfilters

Drei Typen der Anweisungen können unterschieden werden: terminierend, nicht terminierend und weiterleitend bezüglich der Regelevaluation im aktuellen task. Eine terminierende Entscheidung würde das Paket verwerfen und die weitere Untersuchung abbrechen, eine nicht terminierende würde mit den nachfolgenden Regeln fort-

fahren, eine weiterleitende würde den Arbeitsschritt als bearbeitet ansehen und mit dem nächsten task weiter machen. Eine Zuordnung von bekannten Anweisungen und Funktionen zu den Ergebnisarten ist der Tabelle 5.1 zu entnehmen. Abbildung 5.3 veranschaulicht die Wirkung der drei Typen auf die Fortsetzung der Abarbeitung der Arbeitsschritte.

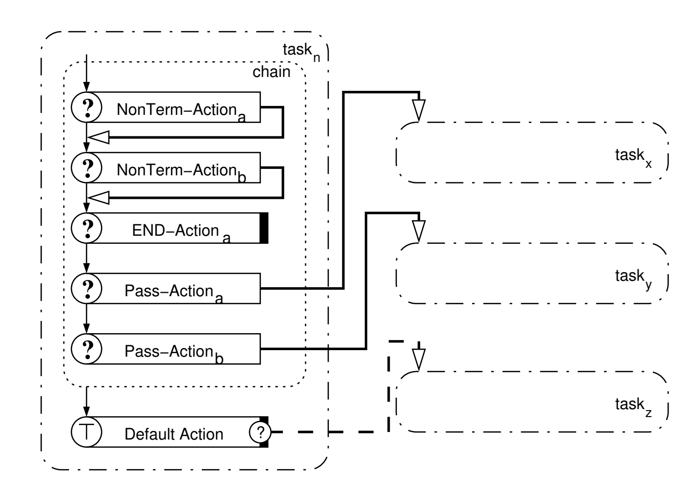


Abbildung 5.3: Ergebnistypen der Firewall-Anweisungen

Genauer betrachtet bestehen viele der Aktionen und Arbeitsschritte der Paketbearbeitung aus einzelnen Schritten, die nicht explizit erkenntlich sind. Ihre Identifizierung hilft aber der Verallgemeinerung der einzelnen Abläufe und der Modellierung.

Unter den impliziten Funktionen wurden folgende Aktionen identifiziert: Regelwerk konsultieren und anwenden; Zustandseintrag anlegen, aktualisieren, konsultieren, anwenden und löschen (nach Timeout) sowie Fragmenten für deren Weiterbehandlung zusammensetzen. Die Zusammensetzung kann entweder nur temporär für die internen Abläufe oder dauerhaft über Systemgrenzen hinweg durchgeführt werden.

Alle vorgestellten Abläufe sind eine direkte oder indirekte Folge aus einer Regel und deren Entscheidung. Dabei wird neben dem Paketfilter auch die dem Routingvorgang zugrunde liegende Routing-Tabelle als Regelwerk verstanden und im folgenden Abschnitt entsprechend modelliert.

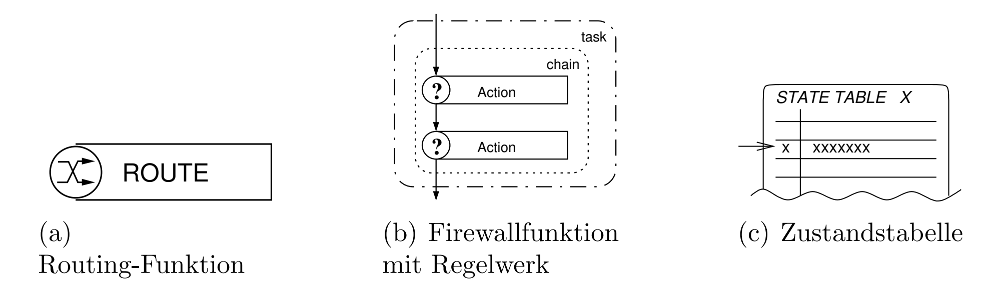
Pakete stellen das Basisobjekt für alle Firewallfunktionen dar. Sie werden oft in Paketverwaltungsblöcken (PVB) verwaltet, die z.B. SocKet Buffer (SKB) bei Linux und memory buffer (mbuf) bei den BSD-Systemen heißen. Sie beschreiben die Eigenschaften des Pakets, enthalten einen Verweis auf das Paket selbst sowie weitere implementierungsspezifische Informationen.

Kontext- und verbindungsorientierte Paketverarbeitung benötigt Zustandsinformationen, um Abhängigkeiten zwischen den einzelnen Paketen herstellen zu können. In Abschnitt 4.2 wurde für die untersuchten Firewalls gezeigt, dass diese Daten sehr unterschiedlich aussehen können. Neben den vordergründigen Verbindungszuständen, können auch Zustände für die aktiven Adressumsetzungen, die durchgeführten Paketmodifikationen und die unvollständigen IP-Fragmente angelegt werden. Hinzu kommen noch statistische Daten oder weitere ergänzende Daten zu den vom Benutzer

definierten Objektgruppen.

Die Daten werden in unterschiedlichen Ausführungsbereichen verarbeitet. Damit sind die ausführenden Instanzen Betriebssystem bzw. Netzwerkstapel, Paketfilter, Anwendungsinspektion und Benutzerprozess gemeint. Dadurch unterscheiden sie sich in ihrer Lebenszeit und ihrer Wirkungsreichweite. Änderungen am PVB sind nur innerhalb des Systems bzw. innerhalb des Kernels gültig und gelten nur für das eine referenzierte Paket. Informationen zum Verbindungszustand sind weiterhin auf das System beschränkt, betreffen aber alle zur Verbindung gehörenden Pakete. Änderungen am Paket sind dauerhaft für alle folgenden Verarbeitungsphasen sichtbar und gelten über Systemgrenzen hinweg. Die Entscheidungen müssen aber für jedes Paket einer Verbindung neu evaluiert werden.

Für Regelwerke, Aktionen, Zustandsdaten, -modifikationen und -abfragen werden im Folgenden Abbildungen eingeführt, die für die symbolische Modellierung aller Arbeitsschritte genutzt werden. Sie ergänzen die bereits in Abbildung 4.1 eingeführten Symbole.



(a) Routing-Funktion

(b) Firewallfunktion mit Regelwerk

(c) Zustandstabelle

Abbildung 5.4: Symbole zur Modellierung der Arbeitsschritte

Zusammengefasst besteht die Modellierung der Firewallsysteme aus Funktionen, Arbeitsobjekten, Zustandsdaten und Zustandsfunktionen, die in Tabelle 5.2 aufgeführt sind.

Funktionen	Paketfilter, Paketmodifikation, NAT, Routing, IDS/IPS, Proxy, Authentifikation, QoS, VPN
Arbeitsobjekte	Pakete, Paketverwaltungsblöcke, Regelwerk(e), Routingtabelle(n)
Zustandsdaten	Assoziationen, Modifikationen, IP-Fragmente, NAT-Einträge, Statistikdaten
Zustandsfunktionen	anlegen, aktualisieren, abfragen, anwenden, löschen

Tabelle 5.2: Arbeitsdaten und -mittel der Firewallsoftware

Durch unterschiedliche Nutzungen der Arbeitsobjekte, der Zustandseinträge und -funktionen lassen sich verschiedene Abläufe bei der Paketverarbeitung festlegen. Durch die Abfrage der bereits vorhandenen Zustandseinträge für das aktuell untersuchte Paket können abhängig vom internen Zustand unterschiedliche Abläufe realisiert werden. So werden zur Optimierung der Rechenzeit und der Laufzeit häufig

die aufwendigeren Vergleiche mit dem Regelwerk übersprungen, wenn ein geeigneter Zustandseintrag vorhanden ist und angewendet werden kann.

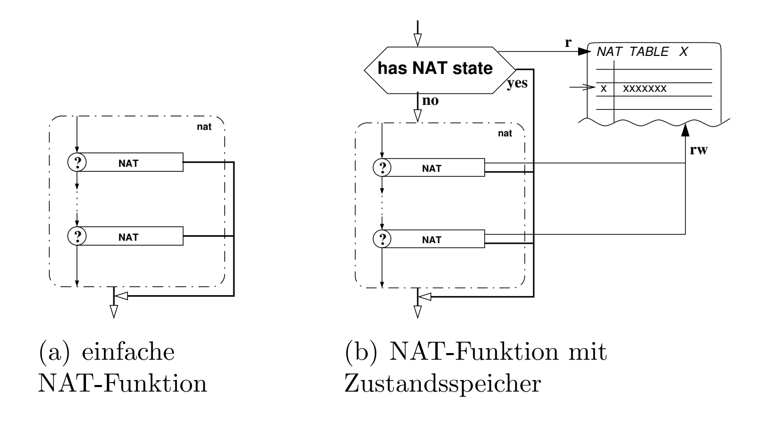
5.2. Anwendung auf Firewallfunktionen

Die in Tabelle 5.2 genannten Funktionen und Arbeitsmittel der Firewalls werden in diesem Abschnitt mit den vereinbarten Symbolen modelliert und besprochen. Aus der Gruppe der Funktionen werden die Paketfilterung, die Paketmodifikation, NAT und Routing behandelt. Die ID&P-Komponenten, Authentifikation, QoS und VPN werden nur als eine nicht näher definierte *Black-Box* im Paketfluss betrachtet, aber aufgrund der Zielstellung dieser Arbeit nicht weiter untersucht.

Die Funktionen Paketfilterung und -modifikationen sind bereits mit der Symboldarstellung in Abbildung 5.4(b) genügend beschrieben. Interessant ist die Kombination dieser Funktionen mit der Verwendung von Zuständen für Assoziationen, Modifikationen, NAT, IP-Fragmente oder Statistikdaten.

Die NAT-Funktion ist ein gutes Beispiel für die Vorteile, die sich aus der Aufspaltung der Regelwerk- und Zustandstabellennutzung in die einzelnen Schritte ergeben. Dadurch ist es möglich, sowohl zustandsbehaftete wie auch zustandslose NAT-Varianten zu modellieren. Letztere haben aber inzwischen an Bedeutung verloren, da sie eine eingeschränkte Funktionalität und weniger Einsatzmöglichkeiten bieten (vgl. die Einführung zu NAT auf Seite 9).

Die zustandslose Variante wird durch die zwei Arbeitsschritte 'Regelwerk konsultieren' und gegebenenfalls 'Regelwerk anwenden' beschrieben. Bei der Verwendung von Zustandseinträgen würden noch die Arbeitsschritte 'Zustand anlegen', 'Zustand konsultieren' und 'Zustand löschen' hinzukommen (vgl. Abbildung 5.5). Dabei würde bei Übereinstimmung 'Zustand konsultieren' auch gleichzeitig das Anwenden der in den Zustandsdaten gespeicherten Transformationen beduten.



(a) einfache NAT-Funktion

(b) NAT-Funktion mit Zustandsspeicher

Abbildung 5.5: Zwei Modellierungen der NAT-Funktion

Ähnlich zu der NAT-Funktion mit Zustandseinträgen kann die Speicherung von Fragmenten mittels *keep frags* bei IPFilter modelliert werden. Hierbei wird relativ

früh im Datenfluss ein Zustandseintrag abgefragt, der beim Evaluieren einer entsprechenden Filterregel für das erste Fragment im neuen Kontext angelegt wurde. Bei einer Übereinstimmung mit dem gespeicherten Zustand wird die erneute Auswertung des Regelwerks übersprungen.

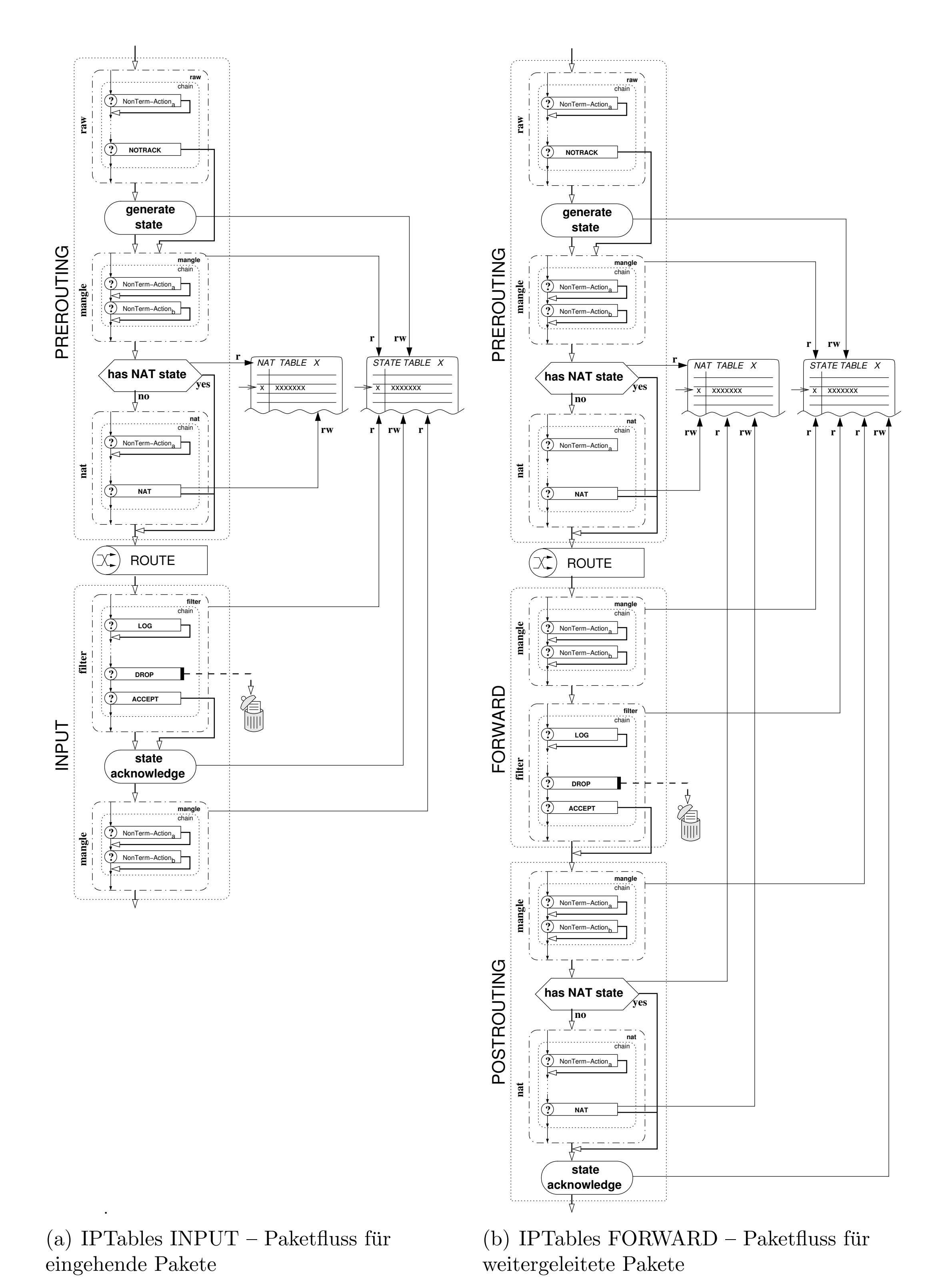
Mithilfe der gewählten Modellierung lassen sich auch die verschiedenen Routingarten zusammenfassen und zu den Firewall-Funktionen aufnehmen. Klassisches Routing wird vom Netzwerkstapel bzw. Kernel erledigt und basiert lediglich auf der Ziel-Adresse. Daneben existieren auch mächtigere Routingvarianten wie policy-based routing (PBR) – also das Routing aufgrund von Quell-Adresse, Ports oder Anwendungsschichtprotokoll – sowie Load-Balancing als Speziallfall des Routings. Sie alle greifen auf eine Routingtabelle oder ein Regelwerk zu, ermitteln den nächsten Vermittlungspunkt sowie den ausgehenden Netzwerkadapter und tragen diese Informationen in den Paketverwaltungsblock ein. Somit ist dieses Vorgehen von den Arbeitsschritten her nicht anders als die bereits vorgestellten Paketmodifikationen und -markierungen. Deswegen können mehrere der untersuchten Firewalls Routingaufgaben unterstützen oder sogar komplett übernehmen. Im Gegensatz zum Kernel-Routing kann die Firewall zahlreiche zusätzliche Entscheidungskriterien zum Routing ermitteln, die auch für ihre Filterfunktion notwendig sind. Damit stellt sie ein mächtiges Routingwerkzeug zur Verfügung. Der vorgegebene Datenfluss einer Firewall kann sich jedoch einschränkend auf die Routing-Möglichkeiten auswirken, wenn das Netzwerkstapel-Routing nicht umgangen werden kann.

5.3. Anwendung auf den Paketfilter netfilter/iptables

In diesem Abschnitt wird die eingeführte Modellierung beispielhaft auf netfilter/iptables angewendet. In Abschnitt 3.3 wurde dieser Paketfilter bereits als funktionell vergleichbar mit den anderen Paketfiltern ermittelt. Aufgrund seiner modularen Struktur und der Möglichkeit alle Regeln explizit zu formulieren lassen sich die Abläufe und Funktionen der anderen Paketfilter nachbilden.

Bei der Nutzung der Zustandseinträge konnten zwei Varianten identifiziert werden: die fest eingestellte Abfrage, die einer Designentscheidung folgt, und die explizite Abfrage durch eine benutzerdefinierte Regel, wie es bei iptables mit den Verbindungseinträgen der Fall ist. Erstere wird von mehreren Firewalls zur Reduzierung der benötigten Abfragen für akzeptierte Assoziationen fest vorgegeben. Hier hat der Benutzer kaum Einfluss auf die Reglementierung der Pakete, die einer bestehenden Assoziation zugeordnet werden. Bei iptables läßt sich sowohl eine frühe Entscheidung für bekannte Assoziation treffen als auch zusätzliche Überprüfungen umsetzen.

In Abbildung 5.6 werden die Paketflüsse für ankommende und für weitergeleitete Pakete dargestellt. Der Paketfluss für ausgehende Pakete wird nicht separat dargestellt, da er im Vergleich zu den beiden gewählten Paketflüssen keine zusätzlichen Abläufe oder Strukturen einführen würde. Die möglichen Aktionen des Paketfilters werden in Tabelle A.1 den einzelnen Phasen sowie der jeweiligen Auswirkung auf den Paketfluss zugeordnet.



(a) IPTables INPUT – Paketfluss für eingehende Pakete

(b) IPTables FORWARD – Paketfluss für weitergeleitete Pakete

Abbildung 5.6: Modellierung der Paketflüsse bei IPTables

6. Testen

In den vorangegangenen Kapiteln wurden repräsentative Firewalls und Paketfilter untersucht, um die verschiedenen Typen von Firewalls und deren Funktionen voneinander abzugrenzen. Damit ist es möglich die zustandsbehafteten Paketfilter zu definieren und eine repräsentative Auswahl der gebotenen Funktionen zu treffen. In diesem und den folgenden Kapiteln wird darauf basierend eine allgemeine Teststrategie für zustandsbehaftete Paketfilter entwickelt. Dafür werden zunächst in diesem Kapitel die Grundlagen des Testens vorgestellt, zu denen eine Definition des Begriffes und die gegenseitige Abgrenzung verschiedener Testarten gehört.

In Abschnitt 6.1 werden die Vorgehensweisen und Grundlagen der Netzwerk- und Protokolltests vorgestellt und in Abschnitt 6.2 wird das Testen auf Firewalls fokussiert. Abschnitt 6.3 behandelt die Testbarkeit von Protokollen und Protokollflüssen und die daraus resultierenden Folgen für die geplannte Teststrategie.

Testen ist ein Oberbegriff für Verfahren, deren Ziel es ist, durch die Überprüfung eines Systems seine charakteristischen Merkmale zu bestätigen, nicht beabsichtigte bzw. nicht erwartete Eigenschaften aufzufinden und sie gegebenenfalls zu beseitigen. Es ist eine Form der Qualitätssicherung, die Fehlverhalten im Betrieb sowie die daraus resultierenden Konsequenzen für Hersteller und Betreiber minimieren und das Vertrauen in ein System stärken kann.

Tests können jedoch nicht die Abwesenheit von Fehlern garantieren, da allumfassendes Testen von komplexen Systemen mit den verfügbaren Technologien und Ressourcen nicht realisierbar ist. Trotz der verbliebenen Restunsicherheit kann auf Tests nicht verzichtet werden, weil das bekannte und spezifizierte Verhalten bestätigt werden kann und der Verzicht auf das Testen keine Alternative darstellt.

Es gibt eine Reihe von verschiedenen Testansätzen, die von informalen ad-hoc bis formal spezifizierten und kontrollierten Methoden reichen. Im Folgenden wird Testen als ein technischer Prozess, der einer spezifizierten und damit wiederholbaren Prozedur folgend in einer kontrollierten Umgebung durchgeführt wird, verstanden.

Das Wasserfallmodell [Roy87], eine Vorgehensweise im Softwareentwicklungsprozess, gibt für jede Phase der Softwareentwicklung Verfahren an, die jeweils andere Fehleigenschaften aufzeigen können und zur Prüfung der Zwischenergebnisse eingesetzt werden, bevor die nächste Phase begonnen wird. Zur Abgrenzung der Begriffe werden hier Validierung, Verifikation und das Debugging nach *Brockhaus Naturwissenschaft und Technik* [BIFABA03] eingeführt.

Die Validierung ist eine Überprüfung der Übereinstimmung von Zielstellung und erreichtem Ergebnis. Bei Software ist damit auch die Gültigkeit eines Objektes in Bezug auf den definierten Wertebereich und Typ gemeint.

Verifikation ist ein formaler Nachweis mit mathematischen Methoden, der die Kor-

rektheit von Systemen und deren Komponenten, d.h. die Übereinstimmung der Isteigenschaften mit den durch die Spezifikation formal beschriebenen Solleigenschaften, bestätigt.

Ein Fehlverhalten, das auf einen internen System-Fehlzustand hindeutet, wird mit Hilfe von Debugging-Werkzeugen untersucht. Debugging ist das schrittweise und protokollierte Ausführen eines Programms zum Auffinden der Fehlerquellen und -ursachen für bereits gesichtetes Fehlverhalten von Software.

Bisher wurde allgemein von Fehlern oder Fehleigenschaften gesprochen. Das Institute of Electrical and Electronics Engineers (IEEE) hat in der Softwaretechnik die Begriffe „error“, „fault“, „failure“ und „mistake“ identifiziert und mit den folgenden Definitionen belegt [IEE90]:

failure / Störung, Fehlerwirkung oder Symptom – Eine Fehlerwirkung bedeutet, dass die Software sich nicht wie erwartet verhält und den spezifizierten Dienst nicht erbringen kann. Beispiel: Programm stürzt aufgrund eines Zugriffs auf einen undefinierten Wert ab.

error / Abweichung oder Fehler – Eine Abweichung ist ein Unterschied zwischen einem berechneten, beobachteten oder gemessenen und dem spezifizierten oder theoretisch korrekten Wert bzw. Zustand. Beispiel: ein Unterschied von 30 Metern zwischen einem berechneten und dem korrekten Resultat.

fault / Defekt, Mangel oder Fehlerzustand – Ein Defekt ist eine inkorrekte Anweisung oder ein Datum, das durch einen Programmierfehler hervorgerufen wird und unter bestimmten Bedingungen zu einer Fehlerwirkung führt. Beispiel: eine falsche Anweisung in einem Computerprogramm.

mistake / Fehlhandlung oder -entscheidung – Eine Fehlhandlung entsteht durch eine unsachgemäße Bedienung des Systems, eine menschliche Tätigkeit, die ein falsches Resultat produziert. Beispiel: eine falsche Eingabe durch einen Benutzer.

Zu jeder Fehlerklasse existieren verschiedene Gegenmaßnahmen, die die Fehler minimieren können. Fehlhandlungen können durch Standards, Vorschriften und Aufklärung bzw. Einweisung vermieden werden. Fehlerzustände und Abweichungen deuten auf interne Defekte im System hin und können durch Debugging aufgespürt werden. Fehlerwirkungen schließlich sind Fehlverhalten, die durch das Testen gefunden werden können.

Mit der Zeit und durch gegebene Anforderungen haben sich zahlreiche Testtechniken etabliert. Sie lassen sich grob nach Testaspekt, Testebene und Testzugang unterscheiden.

Testen als eine Maßnahme der Qualitätssicherung überprüft verschiedene Aspekte der Qualität, die das Testziel festlegen können. Eine grobe Kategorisierung unterscheidet zwischen funktionalen und nicht-funktionalen Eigenschaften. Die ISO/IEC-

Norm[1] 9126 [ISO9126] beschreibt Qualitätsmerkmale für Software und definiert eine feinere Gliederung der Merkmale, die sie den Gruppen

- Funktionalität (functionality)
  - Korrektheit, Angemessenheit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit
- Zuverlässigkeit (reliability)
  - Reife, Fehlertoleranz, Wiederherstellbarkeit, Robustheit, Verfügbarkeit
- Benutzbarkeit (usability)
  - Verständlichkeit, Bedienbarkeit, Erlernbarkeit
- Effizienz (efficiency)
  - Wirtschaftlichkeit, Zeitverhalten, Verbrauchsverhalten
- Wartungsfreundlichkeit (maintainability)
  - Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit
- Übertragbarkeit (portability)
  - Anpassbarkeit, Installierbarkeit, Austauschbarkeit

zuordnet. Zusätzlich zu den genannten Merkmalen kann in jeder der Gruppen die Konformität (conformity) mit der Spezifikation überprüft werden.

Abhängig vom Testaspekt muss eine geeignete Testmethode gewählt werden, aus der sich die Testebene, der Testzugang sowie die benötigte Informationsmenge ergeben. Liggesmeyer [Lig02, S. 34] schlägt eine Klassifizierung vor, die hier verkürzt wiedergegeben wird.

- statisch (Test ohne Programmausführung)
  - verifizierend
  - analysierend
- dynamisch (Test während der Programmausführung)
  - strukturorientiert (Maß für die Überdeckung des ...)
    - Kontrollflusses
    - Datenflusses
  - funktionsorientiert (Test gegen eine Spezifikation)

[1] *International Organization for Standardization* (ISO) und *International Electrotechnical Commission* (IEC)

    * Funktionale Äquivalenzklassenbildung
    * Zustandsbasierter Test
    * Ursache-Wirkungs-Analyse
    * Syntaxtest
    * Transaktionsflussbasierter Test
    * Test auf Basis von Entscheidungstabellen
    * ... und weitere
  - diversifizierend (Vergleich der Testergebnisse mehrerer Versionen)
    * Regressionstest
    * ... und weitere
  - Bereichstest
  - Statistischer Test
  - Error guessing
  - Grenzwertanalyse
  - Zusicherungstechniken.

Aus der gewählten Prüftechnik ergibt sich der notwendige Testzugang, der den Zugangspunkt zur Testebene und die zur Testdurchführung benötigte Informationsmenge beschreibt. Dabei werden im Gegensatz zueinander White-Box- (auch Glass-Box genannt) und Black-Box-Tests unterschieden. Im ersten Fall sind die interne Struktur und der Inhalt bekannt, so dass jeder Ausführungspfad gezielt getestet, inspiziert oder sogar manipuliert werden kann. Diese Methode wird vorzugsweise für Quellcode-Tests benutzt. Bei der Black-Box-Methode können lediglich die Eingabeparameter beeinflusst und die Ausgaben an der externen Schnittstelle eines Systems beobachtet werden. Eine Spezifikation der internen Abläufe oder Strukturen liegt nicht vor. Mischformen zwischen den beiden werden auch als Grey-Box-Tests bezeichnet.

Alle statischen und manche dynamischen Prüftechniken (z.B. alle strukturorientierten Tests) bedürfen des Zugangs zum Programmcode und können als White-Box-Tests klassifiziert werden. Die Black-Box-Testtechniken basieren nicht auf dem Programmcode, sondern auf Spezifikationen. Dazu zählen u.a. alle funktionsorientierten Methoden, deren Testfälle eine vollständige Abdeckung der Spezifikation, aber keine garantierte Vollständigkeit der Abdeckung der Programmstruktur erlauben. Die im Folgenden behandelten Netzwerk- und Protokolltests gehören dieser Kategorie an.

6.1. Netzwerk- und Protokolltests

Das Testen von Protokollen und Netzwerkgeräten setzt auf den zuvor vorgestellten Prinzipien auf und wurde in den Normen ISO/IEC 9646 bzw. ITU-T[2] X.290 [X290]

[2] *International Telecommunication Union (ITU), Telecommunication Standardization Sector* ist aus dem früheren *Comité Consultatif International Téléphonique et Télégraphique* (CCITT) hervorgegangen.

festgehalten. Stellvertretend für beide Normen wird jetzt von ISO9646 gesprochen.

Beide Normen beschreiben ein Rahmenwerk und die Methodologie des Konformitätstestens von Protokollen. Andere Testarten wie Robustheits-, Leistungs- bzw. Belastungstests sind in ISO9646 nur soweit berücksichtigt, wie sie zur Erbringung der Konformität mit der Spezifikation notwendig und ein Teil dieser sind. Das Rahmenwerk enthält Vorschriften und Anleitungen nach denen Protokolle eindeutig formuliert, Test Suiten erstellt und der Prozess des Testens auf Konformität ausgeführt werden sollen. Die einzelnen Aspekte wurden in sieben Teile gegliedert, wobei nur die allgemeinen Konzepte in Teil 1 für diese Arbeit von Bedeutung sind. Der interessierte Leser sei auf die jeweiligen Standards oder [BG92; Bär94] als Sekundärliteratur verwiesen, auf denen die folgenden Abschnitte basieren.

Nach ISO9646 wird ein getestetes System im OSI-Kontext (vgl. Abbildung 2.1) als konform bezeichnet, wenn es den Anforderungen der entsprechenden OSI-Spezifikationen bei der Kommunikation mit anderen Systemen entspricht. Die Anforderungen werden in notwendige, bedingte und optionale Anforderungen untergliedert, die jeweils als Gebote oder Verbote ausgedrückt werden können. Schließlich wird noch zwischen statischen und dynamischen Konformitätsanforderungen unterschieden. Erstere beschreiben die zugelassenen Kombinationen von Fähigkeiten – insbesondere die kleinste Teilmenge, die erforderlich ist, damit die Zusammenarbeit zwischen Systemen möglich ist. Die dynamischen Konformitätsanforderungen machen jeweils den größten Teil einer Protokollspezifikation aus und legen den Gesamtumfang des potentiell möglichen und erlaubten Protokollverhaltens von Implementierungen fest.

Um diese Flexibilität in der Spezifikation von Protokollen kontrollieren zu können müssen zur Planung und Durchführung von Tests Fragebögen vom Testkunden ausgefüllt werden. Das PICS-Formular (protocol implementation conformance statement) gibt an, welche der Anforderungen und Fähigkeiten die zu testende Protokollimplementierung verspricht und dient dazu, relevante Testfälle auszuwählen. Darüber hinaus werden zusätzliche Informationen über die gewählten Parameter und Standardwerte im PIXIT (protocol implementation extra information for testing) angegeben.

Damit die Tests entsprechend den Formularen konsistent und in einer definierten Form ausgewählt werden können, schlägt ISO9646 eine Strukturierung für die Testreihen (test suite) vor, die in Abbildung 6.1 in UML-Notation abgebildet ist.

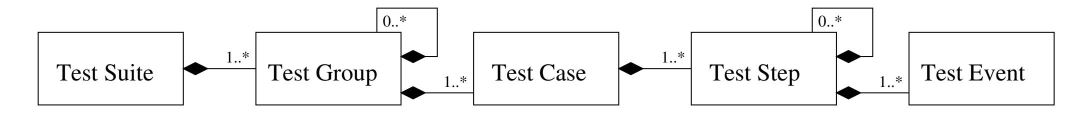


Abbildung 6.1: Struktur einer ISO9646-Testsuite

Weiterhin wird für die Konformitätstests, analog zu den OSI-Protokollen, die keine bestimmte Implementierung vorschreiben, zuerst eine „abstract test suite“ (ATS) spezifiziert, die in einer „(parameterized) executable test suite“ (PETS bzw. ETS) umgesetzt wird. Die ATS ist eine Sammlung von Testdaten und Testfällen, die eine

abstrakte Sicht auf Aspekte wie Kommunikation, Zeit oder Daten haben. Die (P)ETS kann dagegen in einer realen Umgebung ausgeführt werden.

Um die Testfälle in einer realen Umgebung ausführen zu können, bedarf es einer Interaktion zwischen dem Probanden und dem Tester. Im ISO9646-Jargon heißt die Komponente, die Gegenstand des Konformitätstestens ist, „implementation under test“ (IUT) und kann eine oder mehrere Protokolle in derselben OSI-Schicht oder benachbarten Schichten umfassen. Die IUT ist also ein Teil eines „system under test“ (SUT). Der Tester umschließt die IUT-Schichten von oben und unten, weswegen er in die zwei Komponenten „upper tester“ (UT) und „lower tester“ (LT) aufgeteilt ist (vgl. Abbildung 6.2). Die Prüfmethoden der OSI-Konformitätstests sind generell Black-Box-Tests, die nur mit den äußeren Schnittstellen der IUT – den so genannten „points of control and observation“ (PCO) – kommunizieren und so das externe Verhalten testen. Die Interaktion über die PCOs entspricht dem Grundsatz, das Protokollverhalten und nicht eine bestimmte Implementierung zu spezifizieren. Deswegen wird oft die Form von mehr oder weniger abstrakten endlichen Automaten (finite state machine, FSM) gewählt oder sie lässt sich zumindest in solche überführen. Reale Implementierungen lehnen sich dementsprechend eng an die Beschreibung an, auch wenn nicht alle abstrakten Zustände auch real vorhanden sind.

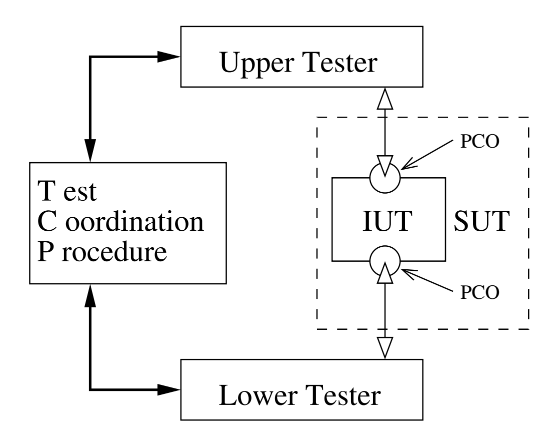
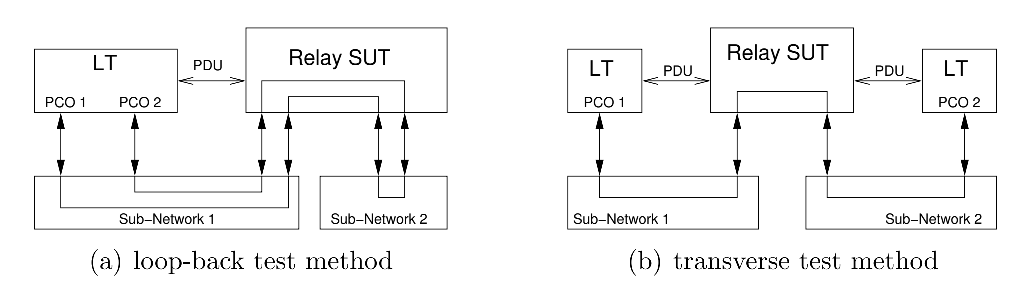


Abbildung 6.2: Konzeptionelle Testarchitektur

Da sich die Implementierungen von Protokollspezifikationen nicht nur in ihrer Konfiguration, sondern auch in Funktion und Aufbau unterscheiden können, wurden sog. abstrakte Testmethoden (abstract test methods, ATM) definiert, die das gesamte Spektrum der zu testenden Systeme abdecken sollen. Unterschieden wird dabei, ob es sich bei dem System um ein Endsystem oder Relaysystem handelt und wie viele der sieben Schichten durch OSI-Protokolle betrieben werden. Daraus ergibt sich der Grad der Kontrollierbarkeit und Einflussnahme auf die IUT.

Anhand der benutzen PCOs sowie der Schnittstelle zwischen LT und IUT bzw. UT und IUT wurden in der Norm verschiedene Testmethoden für vermittelnde und Endsysteme definiert. Für den weiteren Kontext sind allerdings nur Testmethoden für vermittelnde Systeme von Bedeutung. Für IUTs in vermittelnden Systemen gibt es

die „Rückschleifetestmethode“ (loop-back test method) und die „transversale Testmethode“ (transverse test method), die in Abbildung 6.3 abgebildet sind. Diese werden entsprechend der Anzahl der beteiligten Protokollschichten als Single-Party-Testing (SPyT) und Multi-Party-Testing (MPyT) bezeichnet. Jede Protokollschicht eines Endpunktes kommuniziert über „protocol data units“ (PDU) mit der jeweiligen Protokollschicht des Kommunikationspartners. Für die vermittelnden Testmethoden werden nur lower tester eingesetzt, da der Zugriff ausschließlich über die unteren Protokollschichten vermittelt wird und kein Zugriff von oben erfolgt.



(a) loop-back test method (b) transverse test method

Abbildung 6.3: Testmethoden nach ISO/IEC 9646 für vermittelnde Systeme

Da der Quellcode bei Black-Box-Tests unberücksichtigt bleibt, kann ein durchgeführter Test Aussagen liefern, was die IUT falsch gemacht hat, aber nicht wo im Programmcode diese Fehler zu finden sind. Fehlerlokalisierung und Fehlerbehebung werden aus den genannten Gründen in der ISO-Norm 9646 nicht geregelt und bleiben Angelegenheiten des Testkunden bzw. Entwicklers.

6.2. Testen von Firewalls

Netzwerkkomponenten wie Firewalls sind Systeme, die sowohl aus spezieller Software wie auch aus spezieller Hardware bestehen. Das erwartungsgemäße Zusammenspiel dieser Komponenten ist für ihre Zuverlässigkeit, Sicherheit und Leistung verantwortlich.

Dementsprechend kann auch jeder Bestandteil eines Netzwerkgerätes getestet werden. Bei der Hardwareplattform, *firewall hardware* in Abbildung 6.4, können aufgrund der speziell benötigten Messgeräte vor allem durch den Hersteller die physischen Eigenschaften überprüft werden. Auf Bausteinebene können weiter die Schaltungen, die Signalverarbeitung, die implementierte Logik oder die Empfindlichkeit auf Umweltfaktoren getestet werden. Die Betriebssoftware (*firewall software*) kann von den Entwicklern mit den üblichen Mitteln und Werkzeugen der Softwareentwicklung auf Fehler oder Probleme überprüft werden.

Die eigentliche Funktionalität und Dynamik des Systems entsteht erst durch die Konfiguration und Parametrisierung durch den Benutzer. Die Konfiguration heutiger Firewalls ist mit den vielen Eigenschaften und Funktionalitäten sehr komplex geworden. Einfache Regelwerke können bereits über 100 Regeln enthalten und im professionellen Bereich sind 10.000 Regeln auch nicht ungewöhnlich. Die Zuverlässigkeit,

Konsistenz und Widerspruchsfreiheit dieser Konfigurationen nachzuvollziehen und zu gewährleisten ist häufig nur mit Einsatz von unterstützenden Werkzeugen möglich. In Abbildung 6.4 wird zwischen produktgebundener Konfiguration (*fw-specific rules*) und einer abstrakten produktübergreifenden Form (*abstract fw-rules*) unterschieden.

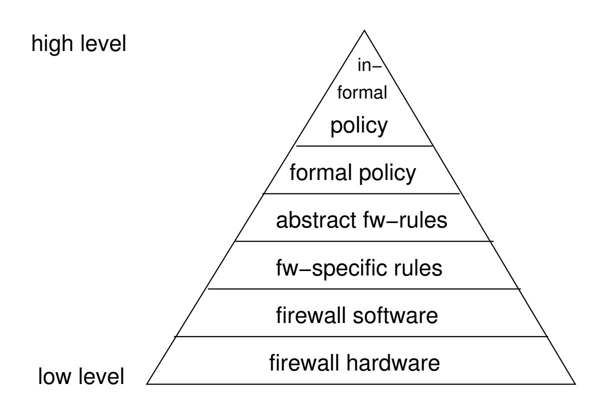


Abbildung 6.4: Abstraktionsebenen beim Testen von Firewalls

Die Konfiguration kann aber nicht nur allein betrachtet werden, sondern soll die Sicherheitsrichtlinien umsetzen. Die Konsistenztests einer Konfiguration sagen aber noch nichts aus über die Übereinstimmung mit den Richtlinien. Diese zu überprüfen ist allerdings bis jetzt nicht so einfach, da sie meist informeller Natur sind und sprachlich festgehalten werden (*informal policy*). Deswegen bedürfte es einer weiteren formaleren Form (*formal policy*) mit festgelegter Syntax und Semantik (vgl. Definition der Sicherheitsrichtlinien in Abschnitt 2.2).

Beim Testen von Netzwerkkomponenten können die zu testenden Aspekte nach verschiedenen Kriterien geordnet werden. Zunächst sei die Rolle des Testers zu betrachten, wobei zumindest Entwickler, Administratoren, Netzwerknutzer und Angreifer unterschieden werden können. Ein Entwickler – sowohl der Hardwareplattform wie auch der Software – überprüft zum Beispiel die Schnittstellen und Abhängigkeiten zu benachbarten Komponenten, die implementierte Funktionalität, vollständige Behandlung möglicher Parameter oder Leistungsmerkmale. Ein Administrator kann zum Teil auch die gleichen Aspekte testen, will aber zudem sicherstellen, dass das System erwartungsgemäß und zuverlässig in der eingesetzten Infrastruktur und im Kontext funktioniert. Netzwerknutzer könnten gezwungen werden Tests durchzuführen, wenn sie im Rahmen der Fehlersuche für sich oder den zuständigen Administrator Fehlverhalten diagnostizieren. Dies könnten gestörte oder eingeschränkte Konnektivität sowie unzureichende Leistung sein. Zuletzt wären noch der sicherheitskritische Administrator oder ein böswilliger Nutzer anzuführen, die sich wie ein Eindringling verhalten. Dieser würde vor allem die Robustheit und (Un-)Zuverlässigkeit eines Systems untersuchen wollen, indem er es grenzwertigen und undefinierten Situationen aussetzt, um damit Fehlverhalten zu provozieren.

Speziell bei Protokoll- bzw. Netzwerktests werden die Quelle der Testdaten und die Art des Interaktion mit dem System unter Test (SUT) unterschieden. Werden die Testdaten live an einem System gewonnen, spricht man von einem „online-Test“.

Wurden die Daten zuvor gesammelt und vom System getrennt in einer Simulation benutzt, so handelt es sich um einen „offline-Test“. Beim online-Test wird noch zwischen der passiven und der aktiven Variante unterschieden. Bei der passiven Variante wird das Verhalten lediglich beobachtet, während bei der aktiven Variante das SUT stimuliert wird, um gewünschte Testdaten zu erhalten. Die oft genannten Abtastungs- und Penetrationstests würden so in die Kategorie aktive online-Tests (bzw. dynamischer, funktionsorientierter Black-Box-Test) eingeordnet werden.

6.3. Testbarkeit von Protokollen und Protokollflüssen

Aufgrund der diskreten Natur und der beschränkten Länge eines Datenpakets sind alle Eigenschaften, die darin abgebildet werden, endlich und bei der zustandslosen Betrachtung als Testeingaben verwendbar, indem alle Wertmuster erstellt und getestet werden. Protokolle basieren aber auf dem Austausch zahlreicher Pakete, die zusammen einen Zustand im Kontext des Paketflusses beschreiben. Für jede Schicht verwaltet das jeweilige Protokoll seinen eigenen Kontext und die dazugehörenden Zustände. Betrachtet man einen realen Netzwerkknoten, der viele Assoziation gleichzeitig vermittelt, so hat jede der Protokollschichten für jede Assoziation jeweils einen Zustand, die alle zusammen wiederum den Gesamtzustand des Knotens darstellen. Für einen vollständigen Test von so einem Knoten, in diesem Fall der Firewall, müssten alle möglichen Zustände überprüft werden. Das würde bereits bei zwei oder drei betrachteten Schichten zu einer Zustandsexplosion führen, da die Potenzmenge einer beliebigen Anzahl von gehaltenen Assoziationen und einer beliebigen Vorgeschichte zu jedem Zustand berücksichtigt werden müsste. Abbildung 6.5 versucht die Zustandsexplosion zu visualisieren.

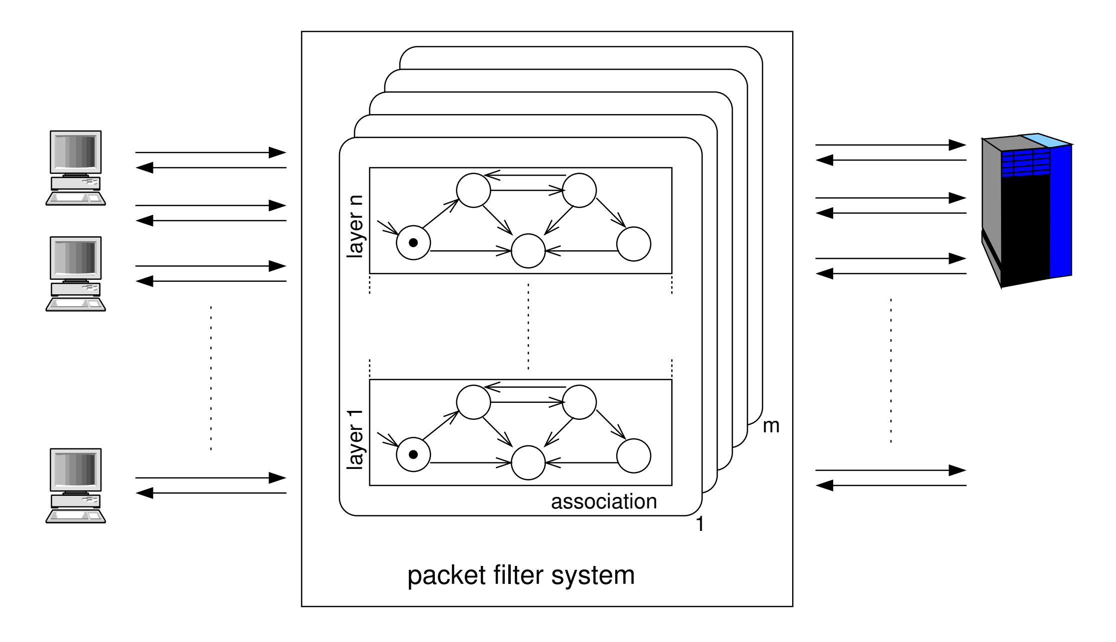


Abbildung 6.5: Problem der Zustandsexplosion

Aus diesem Grund baut diese Arbeit auf der Annahme auf, dass Netzwerk- und Vermittlungsgeräte dafür vorgesehen sind, eine große Menge von beliebigen Datenströmen gleichzeitig sowie ohne gegenseitige Beeinflussung zu verarbeiten und zu transportieren. Durch eine Beschränkung der betrachteten Testeigenschaften und durch geeignete Teststrategien wird das Risiko der gegenseitigen Beeinflussung reduziert. Unvorhergesehene Nebeneffekte können bei fehlerhaften oder nicht dokumentierten Verhaltensweisen jedoch auftreten.

Im Top-Down-Verfahren wird von einem maximal-vollständigen Test ausgegangen und durch schrittweise Einschränkungen ein Optimum für die Balance zwischen Testabdeckung und Ressourcennutzung entwickelt. Dabei werden alle Abweichungen diskutiert und begründet.

Die vollständige Abdeckung des Testraums würde einen Maximaltest erfordern, der alle Variationen der Aspekte für jede der Testeingaben berücksichtigt. Dazu sind für einen zustandsbehafteten Paketfilter zumindest Variationen

- der Länge eines Paketes,
- der Anzahl von Paketen,
- der Abfolge von Paketen,
- des Inhalts eines Paketes,
- des Zeitablaufs zwischen den Paketen

möglich, wobei die Aufzählung nur einen Teil der möglichen Eigenschaften ausmacht. Dies würde bei der Untersuchung eines einzigen TCP/IPv4 Paketes ohne Nutzlast eine Paketlänge von mindestens 40 byte[3] erfordern, was 2^{320} mögliche Bitkombinationen und damit auch Testfälle ergäbe. Durch die Betrachtung eines zustandsbehafteten Testobjektes, das die Pakete in Kontext zueinander stellt, würde die Kombination dieser Eigenschaften zu einer Zustandsexplosion führen, die einen Test mit endlich zur Verfügung stehenden Ressourcen (z.B. Zeit oder Speicher) unmöglich macht.

Um trotzdem einen Test durchführen zu können müssen Möglichkeiten gesucht werden den Testaufwand und die benötigten Ressourcen zu reduzieren. Dabei soll untersucht werden, ob der Testraum bzw. die Testgranularität reduziert werden können, ob alle Variationen auch tatsächlich an dem Testgerät getestet werden können, ob zustandsbehaftete Testfälle aufeinander aufbauen können statt jeden Testfall einzeln zu betrachten und ob bestimmte Tests parallel ausgeführt werden können, um Arbeitspausen zu vermeiden.

Die Aufgabenstellung bezieht sich auf die Überprüfung von spezifikationskonformen Protokollflüssen unter Realbedingungen, wodurch die möglichen Variationen eines Paketes reduziert werden können, da die Anwendung der Spezifikationen die

[3] Die genannten 40 byte ergeben sich aus den Mindeslängen der TCP und IPv4-Header, die jeweils 20 byte betragen. Bei Berücksichtigung der weiteren Nutzlast könnte die Länge bis auf 1500 byte erweitert werden, was der maximale Länge eines über Ethernet verschickten Datagrams entspricht.

Pakete mit einer Semantik belegt, die eingehalten werden muss. Bei Berücksichtigung der Spezifikationen ist es nicht sinnvoll alle möglichen Bitmuster tatsächlich zu testen. Werte außerhalb der Spezifikation, die dazu führen können, dass das Paket nicht interpretiert werden kann, können nur direkt am Probanden über entsprechende Stimulatoren eingegeben werden. Andernfalls würden sie von dazwischen liegenden vermittelnden Knoten oder von den tieferen Protokollschichten im Probanden verworfen werden.

Leider bekommen im Gegenzug die Paketflüsse eine höhere Bedeutung, wobei die Pakete im Kontext zu einander betrachtet werden müssen. Daneben basieren alle Protokolle auf Assoziationen zwischen zwei (oder mehreren) Kommunikationspartnern, weswegen die Adressierung der Partner und die dadurch entstehenden Kombinationen betrachtet werden müssen.

Eine weitere Reduzierung des Testraums bringt die Auswertung der Konfigurationssprachen der betrachteten Paketfilter. Die in Tabelle 3.2 beschriebenen Filterkriterien geben einen Überblick über die Unterscheidungsmerkmale der Paketfilter wieder, weshalb die Testgranularität nur so genau sein muss, wie es in der Konfiguration eingestellt werden kann und der Filter eine Unterscheidung macht.

Schließlich können die Modellierung der internen Paketflüsse und die Erkenntnisse aus den gesammelten Daten zur Zustandsverfolgung dazu benutzt werden, den zeitlichen Rahmen und die erlaubten Paketfolgen zu definieren. Sie werden auch zur Entwicklung der Testfälle und der Strategie bei der Paketauswahl benutzt, um die Laufzeit zu optimieren und Wiederholbarkeit zu erreichen.

6.4. Reduzierung des Testraums

Der Testraum der zustandsbehafteten Filterung ist trotz der hier angeführten Reduzierungen immer noch zu groß, um vollständig getestet zu werden. Deswegen wird eine Strategie eingeführt, mit der die Menge der unterschiedlichen Testpunkte reduziert werden kann. Dazu wird der gesamte Testraum anhand der Unterscheidungsmerkmale der Konfigurationssprache in kleinere Bereiche zerteilt, die jeweils alle Repräsentanten einer Konfigurationseinstellung zusammenfassen. Diese Repräsentanten bilden sogenannte Äquivalenzklassen von denen jeweils nur wenige überprüft werden müssen, um eine Aussage über den Bereich treffen zu können.

Das Konzept der Zerteilung des Testraums in kleinere Bereiche geht davon aus, dass Abweichungen zwischen den Ergebnissen wahrscheinlicher an den Stellen sind, an denen die Firewall aufgrund der Konfiguration oder ihrem Design Entscheidungen trifft und Verzweigungen macht. Im Gegenzug wird erwartet, dass sie sich in stetigen Bereichen auch gleichartig verhält und die Aktionen für Randwerte identisch mit den inneren Werten sind. Eine ähnliche Vorgehensweise wurde auch in „Policy segmentation for intelligent firewall testing“ [IHAS05] verwendet, wobei dort nur der Adressraum zerteilt wurde, der mit verschiedenen aus den Regeln hervorgegangenen Größen gewichtet wurde. Die Autoren zeigen, dass die Auswahl der Testpunkte aufgrund der Segmentierung und der Gewichtung, eine signifikante Verbesserung der

Testabdeckung im Gegensatz zum unsystematischen bzw. zufälligen Testen erreicht. In der vorliegenden Arbeit wird die Segmentierung konsequent auf alle durch die Konfiguration vorgegebenen Filterkriterien angewendet, was zu mehr Testfällen führt, als die Segmentierung in der zitierten Arbeit. Dadurch ist trotz der selektiven Tests in den Grenzbereichen der Segmente eine hohe Testabdeckung möglich.

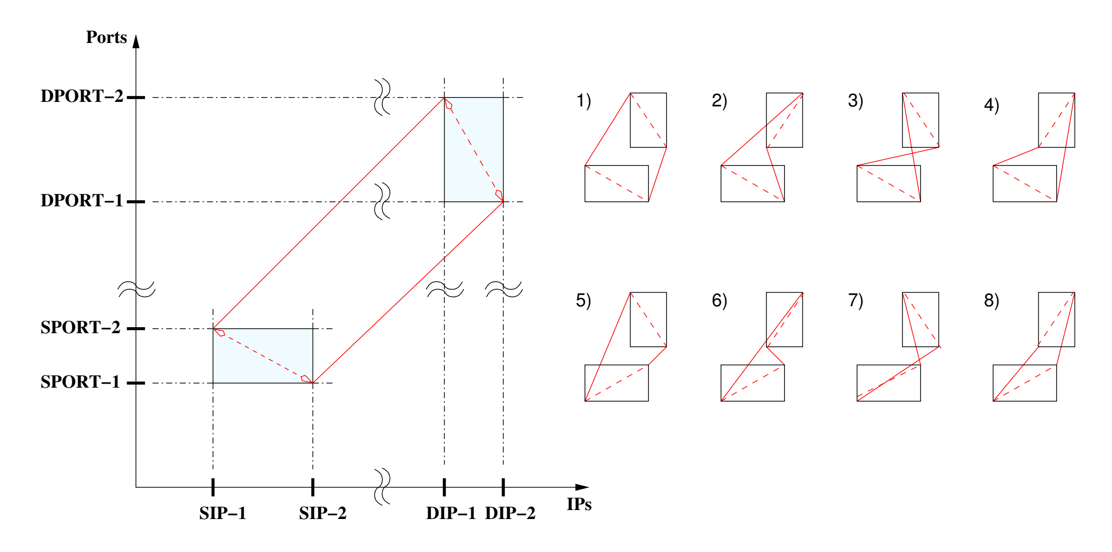


Abbildung 6.6: Diagonale Wahl der IP-Adressen und Ports

Nicht alle möglichen Kombinationen der Randpunkte müssen getestet werden. Für die Überprüfung eines Segmentes müssen jeweils die zwei begrenzenden Punkte, Minimum und Maximum, aus jedem Aspekt gewählt werden, wodurch auch zwei Testfälle bzw. Testpunkte pro Segment entstehen. Betrachtet man die Testpunkte im geometrischen Raum, so würden sie zwei jeweils diagonal zueinander liegende Endpunkte beschreiben.

Bei n betrachteten Aspekten ergeben sich so 2^{n-1} Möglichkeiten für die Wahl der gegenüberliegenden Testpunkte. Die Diagonalen werden ungerichtet betrachtet, weswegen Testpunkt1→Testpunkt2 gleichbedeutend mit Testpunkt2→Testpunkt1 ist. Für das in Abbildung 6.6 angegebene Beispiel mit den vier Aspekten Quell- und Zieladresse sowie Quell- und Zielport ergeben sich $2^{4-1} = 2^3 = 8$ Möglichkeiten die Testpunkte zu wählen.

Sollte eine Firewall neben den identifizierten Segmenten weitere Unterscheidungen treffen, die weder aus der Spezifikation noch aus der Konfiguration ersichtlich sind, so können diese durch die hier entworfene Strategie nicht gefunden werden. Zum Auffinden dieser Art von Unterscheidungen wäre der Zugriff auf den Quellcode notwendig und Debugging statt Testen die geeignetere Methode.

7. Allgemeine Teststrategie

In diesem Kapitel wird ein Konzept für eine allgemeine Teststrategie entwickelt. Dabei werden Arbeit und Ergebnisse der vorangegangenen Abschnitte verwendet. In Kapitel 3 wurden Merkmale und Funktionen der Firewalls identifiziert, die in Kapitel 5 dazu verwendet wurden, ein Modell der jeweiligen Paketflüsse zu erstellen. In Kapitel 6 wurden Grundlagen und die Methodologie des Testens vorgestellt, die nun dazu verwendet werden, eine Teststrategie zu entwickeln und sie in die bekannten Techniken einzuordnen. Für die Interaktion mit der Firewall werden insbesondere die in Abschnitt 4.2 erörterten Mechanismen der Zustandsverfolgung bei der Testdurchführung verwendet.

Ziel der Teststrategie ist, das Verhalten eines typischen zustandsbehafteten Paketfilters in Bezug auf seine Konfiguration, seine Voreinstellungen, die Produktspezifikation bzw. -dokumentation sowie die Protokollspezifikationen zu überprüfen und eventuell vorhandene Abweichungen festzustellen. Die Ergebnisse sollen dazu genutzt werden die Modellierung und die Dokumentation mit dem Realsystem abzugleichen sowie zustandsbehaftete Paketfilter vergleichbar zu machen.

Die gesetzte Aufgabenstellung stellt folgende Anforderungen an die Teststrategie:

- Unabhängigkeit von konkreten Produkten
- Unabhängigkeit von den betrachteten Protokollen
- Einstellbarkeit der Tests für eine konkrete Firewall
- Bewertung der Relevanz und der Detailtiefe der Testfälle
- Optimierung der Nutzung der Testressourcen
- Durchführbarkeit der Tests unter Labor- und Realbedingungen
- Wiederholbarkeit der Tests.

Die Teststrategie wird ohne Einschränkungen auf einen konkreten Paketfilter entwickelt. Sie soll auf beliebige Protokolle anwendbar sein, auch wenn die betrachteten Systeme vor allem die Protokolle der IP-Protokollsuite unterstützen. Zusätzlich wird durch die Betrachtung von mehreren repräsentativen Produkten und die Bewertung bzw. Abgrenzung der Relevanz einzelner Funktionen eine hohe Abdeckung von im Einsatz befindlichen Systemen angenommen.

Die Detailtiefe und die Mächtigkeit der Testfälle werden im Kontrast zu der Ressourcennutzung sowie ihrer Aussagekraft betrachtet und bewertet. Dabei muss für die Verifizierung der spezifikationskonformen Funktion eine realistische Balance zwischen

einem reduzierten und einem vollständigen Test gefunden werden. Gewählte Abweichungen oder Einschränkungen werden kenntlich gemacht, um eine Einschätzung der Ergebnisqualität zu ermöglichen.

Optimierungen sollen identifiziert und die Auswirkungen auf die Testergebnisse aufgezeigt werden. Ziel ist es, den Test auf einfache Protokollflüsse zu reduzieren, so dass sie jeder spezifikationskonforme Paketfilter akzeptiert.

Die Tests sollen unter Labor- und Realbedingungen, also nicht nur in einer Laborumgebung mit direktem bzw. exklusivem Anschluss an den Probanden durchführbar sein.

Die Umsetzung der Teststrategie benötigt Informationen, auf deren Grundlage die Planung und Steuerung der Tests durchgeführt werden kann. Wie in Abbildung 7.1 dargestellt, sind das eine Beschreibung des Probanden, dessen Konfiguration, die zu untersuchenden Testvektoren und die Protokollspezifikationen, mit denen der Test umgesetzt wird. Die Teststrategie steuert den Testablauf, stimuliert den Probanden und wertet die Ergebnisse aus.

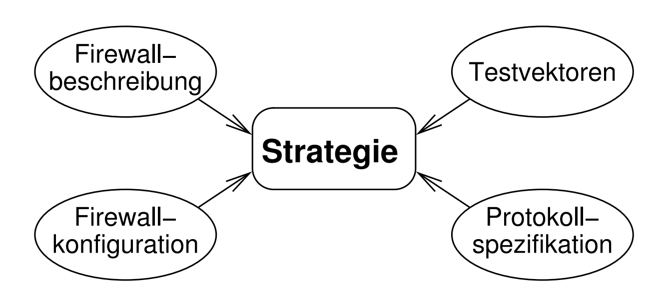


Abbildung 7.1: Benötigte Informationen für die Teststrategie

Zur Beschreibung des Probanden gehören die produktspezifischen Eigenschaften. Dazu zählen die internen Mechanismen bei der Abarbeitung der Pakete, evtl. voreingestellte Standardeinstellungen und implizite Filterfunktionen oder die verwendete Zustandsverfolgung mit ihren Timeouts. Basis der Firewallbeschreibung bildet das entwickelte Modell, welches abstrakte Funktionsbausteine verwendet, die in Firewallsystemen und Paketfiltern identifiziert wurden und mit denen sich durch eine geeignete Kombination verschiedene Produkte abbilden lassen. Das Modell hilft den internen Paketfluss und die Zustandsverfolgung des Probanden zu erfassen, welche für die Steuerung der Tests verwendet werden. Für eine funktionierende Nachbildung der Zustandsverfolgung müssen weiterhin die Zustandsmaschinen der unterstützten Protokolle abgebildet werden, um verschiedene Phasen der Assoziationen gezielt erzeugen zu können. Dabei müssen auch die Timeouts für jede der Phasen berücksichtigt werden, um eine gegenseitige Beeinflussung der einzelnen Tests auszuschließen und auch die Wiederholbarkeit der Tests zu ermöglichen. Für die Vorbereitung und die Auswertung der Tests müssen die unterschiedlichen Arten der Paketfilterung und -veränderung des Testobjektes nachgebaut werden.

Die Firewallkonfiguration umfasst benutzerspezifische Einstellungen des Systems wie einstellbare Parameter und Regeln oder die verwendeten Netzwerkeinstellungen.

Aus der eingestellten Konfiguration bzw. dem Regelwerk ergeben sich die zu untersuchenden Testpunkte, an denen das Verhalten des Probanden überprüft wird. Für die Konfigurationsanalyse soll laut Aufgabenstellung der Firewall-Analyzer (FWA) verwendet werden, da er bereits im Vorfeld dieser Arbeit, mit dem Ziel diese Untersuchung zu unterstützen, konzipiert und implementiert wurde. Der FWA kann ein Firewall-Regelwerk analysieren und in eine produktunabhängige Darstellung überführen. Dabei zerteilt der FWA die im Regelwerk beschriebenen Filterkriterien und Aktionen in disjunkte Bereiche, die in sogenannten Vektoren beschrieben werden. Die Teststrategie benutzt die Testvektoren zur Bestimmung der Testpunkte, für die die Testsequenzen erstellt werden. Aktuell kann vom FWA nur die Konfiguration von netfilter/iptables eingelesen werden, aber eine Erweiterung auf weitere Konfigurationssprachen ist geplant, wodurch auch die Teststrategie für den Test anderer Paketfilter verwendet werden kann.

Für die Erstellung der Testsequenzen und zur Überprüfung des spezifikationskonformen Verhaltens des Probanden werden die jeweiligen Protokollspezifikationen zu Grunde gelegt. Diese werden als Teil der Teststrategie implementiert.

Als Werkzeug für die Erstellung, die Manipulation, den Versand und den Empfang von Paketen wurde FWTest (TU-Berlin) vorgegeben. Es besteht aus einem Steuerprogramm und zwei Agenten, die für den Versand und den Empfang von Paketen zuständig sind. Das Konzept der Agenten ist besonders dazu geeignet, die Firewall von zwei Seiten zu stimulieren bzw. zu beobachten. Die bereits vorhandene Python-Schnittstelle wird jedoch mit Scapy erweitert, das eine mächtigere Funktionalität zum Erstellen und Analysieren von Paketen besitzt.

Testmethodologie und verwendete Bezeichnungen

Mit den Begriffen der Testmethodologie ausgedrückt handelt es sich bei der geplanten Teststrategie um die Überprüfung des Qualitätsmerkmals *Konformität* aus der Kategorie *Funktionalität*. Dem werden die funktionsorientierten Testarten aus der Kategorie der dynamischen Tests zugeordnet. Weiterhin ist es auch möglich diversifizierende Tests durchzuführen, um mehrere Paketfilterversionen zu vergleichen. Bei beiden Arten können Abweichungen und Störungen bzw. Symptome ermittelt werden. Ein hier angestrebter Test des extern beobachtbaren Systemverhaltens gehört den Black-Box-Tests an. Da das Verhalten jedoch unter Berücksichtigung der Konfiguration und einer Modellierung des internen Paketflusses überprüft wird, muss der Test schließlich den Grey-Box-Tests zugeordnet werden. Die Testebene kann als Systemtest klassifiziert werden.

Für die Beschreibung des Konzeptes der Teststrategie sollen die im Folgenden verwendeten Begriffe eingeführt und definiert werden. In Abschnitt 6.1 wurden bereits die Einheiten einer ISO9646-Testsuite eingeführt, die in Tabelle 7.1 den weiter verwendeten Begriffen gegenüber gestellt werden.

Die Teststeuerung, die die Strategie umsetzt, wird unter der Bezeichnung **FWT-Strategy** entwickelt. Im ISO9646-Kontext stellt sie eine *abstract testsuite* dar, die zusammen mit der Parametrisierung durch die Firewallbeschreibung und die Test-

ISO9646-Einheit	Einheit in FWTStrategy
ATS	FWTStrategy
PETS	FWTStrategy mit Firewallbeschreibung und Vektoren
test group	Vektor
test case	Testfall
test step	Schritte im Testpfad
test event	empfangenes Paket oder Zeitüberschreitung
LT	FWTest-Agenten
TCP	Teststeuerung

Tabelle 7.1: FWTStrategy im Kontext von ISO9646

vektoren eine ausführbare Testsuite (*executable testsuite*) ergibt. Aus der Sicht der Testmethodologie setzt FWTStrategy die *transverse test method* für ein *multi-party system under test* um. Die Teststeuerung entspricht der *test control procedure*, die zwei *lower tester* ansteuert (vgl. Abbildung 6.3).

Jedes Unterscheidungsmerkmal aus der Sprache des Firewallregelwerks spannt eine Achse in einem mehrdimensionalen Raum auf, der den Wirkungsraum der Firewall beschreibt. Dieser wird weiter als **Testraum** bezeichnet. Die Unterscheidungsmerkmale entsprechen den durch eine Firewall untersuchten Filterkriterien bzw. der postulierten Firewallaktion. Der Definitionsbereich jeder Achse wird durch die möglichen, diskreten Werte des jeweiligen Unterscheidungsmerkmals definiert. Der Testraum oder Ausschnitte davon können durch einen **Vektor** beschrieben werden, dessen Koordinaten durch die Unterscheidungsmerkmale vorgegeben sind. Die hier betrachteten Vektoren beschreiben disjunkte Teilräume des gesamten Testraums, indem sie die Koordinaten als Tupel der Grenzen eines stetigen Abschnitts innerhalb der jeweiligen Achse angegeben. Die aufsummierten Vektoren ergeben wieder den gesamten Testraum. Ein Vektor mit einem eindeutigen Wert für jede Koordinate definiert einen **Testpunkt** im Testraum.

Aus einem Testvektor kann eine Vielzahl von Testpunkten abgeleitet werden. Getestet wird jedoch nur eine Auswahl der Testpunkte, die die Randwerte des jeweiligen Hyperrechtecks beschreiben. Für die Teststrategie werden aber nicht Testpunkte, sondern **Testfälle** aus den Vektoren erstellt. Testfälle können als ein teilweise fest belegter Testvektor verstanden werden, der noch einige Variablen enthält. Ein Testfall dient als Schablone für die Herstellung verschiedener Pakete eines Protokolls aus dem Testvektor unter Berücksichtigung des gewählten Testpunktes. Für jedes im Testvektor beschriebene Protokoll und für jeden unterschiedenen Assoziationszustand wird ein Testfall vorbelegt. Sind schließlich alle Aspekte mit eindeutigen Werten belegt beschreibt ein Testfall einen eindeutigen Testpunkt, der in ein **Testpaket** überführt werden kann.

Mehrere solcher Testpakete können zu einem logischen **Testpfad** angeordnet werden, um einen kontextabhängigen Zustand der Firewall zum Zweck der zustandsbehafteten Filterung testen zu können. Die vorangegangenen Testpakete in einem Kontext werden als **Vorgeschichte** für die nachfolgenden bezeichnet. Dieser Kontext

kann entweder über protokollabhängige Pakete ausgelöst oder nach Überschreitung einer Firewall-abhängigen Zeitschranke (engl. **Timeout**) automatisch erreicht werden.

Das Konzept der Teststrategie in Abbildung 7.2 fasst die bisher eingeführten Begriffe und Komponenten zusammen.

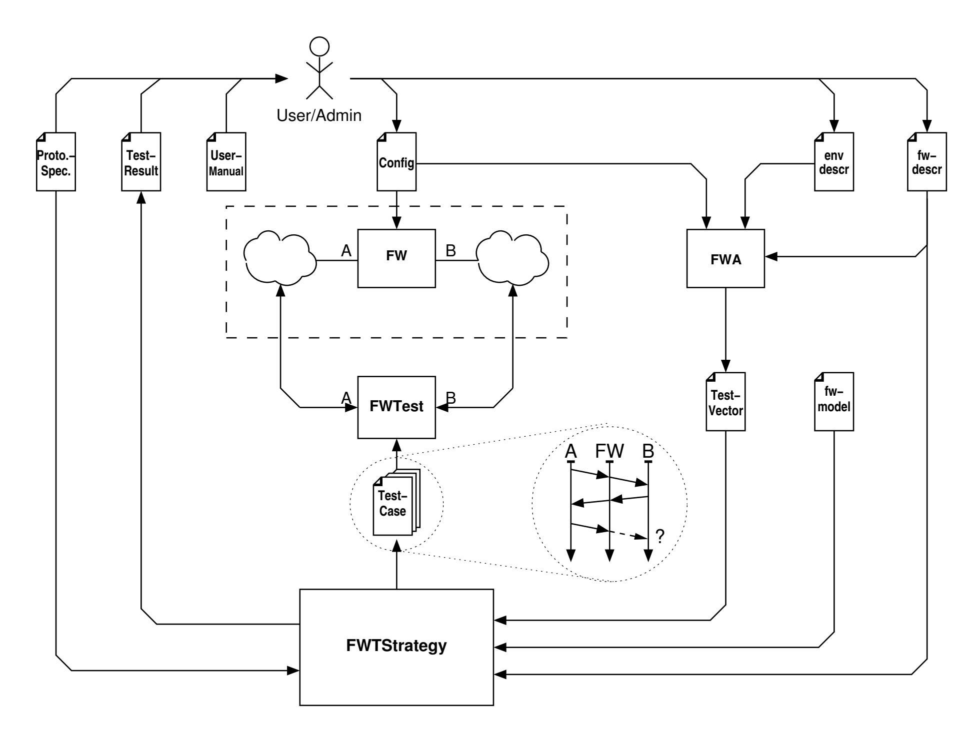


Abbildung 7.2: Interaktion der Komponenten in FWTStrategy

7.1. Vereinfachter Testablauf für einen Testpunkt

In diesem Abschnitt wird die Teststrategie entwickelt. Zuerst wird ein vereinfachter Testablauf diskutiert, der einen Test für einen einzigen Testpunkt durchführt. Dieses Vorgehen wird analysiert, um das Prinzip eines solchen Tests in Abschnitt 7.2 auf eine ganze Testreihe zu übertragen. Dabei werden die Probleme einer direkten Übertragung aufgezeigt, um im folgenden Abschnitt die gewählte Lösung mit den Optimierungen vorzustellen. Die konkrete Umsetzung der dort behandelten Algorithmen wird in Kapitel 8 besprochen.

Ausgehend von der Abbildung 7.1 kann die Teststrategie in mehrere Schritte unterteilt werden, in denen die Aufgaben

- Eingabe eines Testpunktes,
- Einstellung der Teststrategie auf den Probanden,
- Vorbereitung des Probanden für den Test,

- Durchführung und Auswertung des Tests und
- Bericht an den Benutzer

gelöst werden müssen.

Der FWA übernimmt die Analyse des Regelwerkes und gibt der Teststrategie die zu testenden Bereiche vor. In der weiteren Betrachtung wird nur von einem Vektor ausgegangen, der den zu testenden Bereich beschreibt. Tabelle 7.2 zeigt eine vereinfachte Darstellung der in einem Vektor beschriebenen Angaben. Eine vollständige Beschreibung der Vektoren und der Schnittstelle zum FWA wird im Anhang in Abschnitt A.2 dokumentiert.

Eigenschaft	**Werte(bereich)**
Protokoll(e)	TCP, UDP
Senderichtung	A-zu-B
Quelladresse	10.0.0.0 – 10.0.0.255
Zieladresse	10.1.0.0 – 10.0.0.255
Quellport	1025 – 32000
Zielport	80
Zustände	new, established
FW-Aktionen	log, masquerade-SNAT, accept

Tabelle 7.2: Beispiel für die in einem Vektor beschriebenen Informationen

Neben den Vektoren liefert der FWA eine Angabe des zu testenden Paketfilters, also z.B. netfilter/iptables. Damit kann die Teststrategie die dazu passende Beschreibung des Probanden laden und den Vektor interpretieren.

Der FWA arbeitet (fast) vollständig auf der syntaktischen Ebene der Konfiguration und macht deswegen keine Aussagen oder Annahmen über die Semantik der einzelnen Symbole. Dementsprechend unterscheidet der FWA genauso viele Symbole und Werte wie die jeweilige Konfigurationssprache vorsieht. Zu den grundlegenden Ausnahmen gehören die Unterscheidung und die Auswirkung von terminierenden Anweisungen wie `ACCEPT` und `DROP`. Die Belegung der Symbole mit einer Semantik wird erst durch die Anwendung der Spezifikationen und der Probandenbeschreibung in FWTStrategy durchgeführt. Zum Beispiel muss bei der Interpretation der ausgegebenen IP-Adressen gegebenenfalls eine Anpassung durchgeführt werden, da der FWA die Netzwerkbereiche aus der Konfiguration rein mathematisch interpretiert. Er liefert z.B. für das Teilnetz 10.0.0.0/24 die Eckpunkte 10.0.0.0 und 10.0.0.255, die wahrscheinlich aus der Sicht eines Routers spezielle Adressen (Broadcasts) sind, die unter Umständen nicht weitergeleitet werden. Die Interpretation von Netzmasken und Netzsegmenten muss in der Teststeuerung durchgeführt werden, wobei der Benutzer über jede Abweichung von den Segmenträndern informiert werden muss.

Ein vollständiger Test dieses Vektors würde $255 \times 255 \times 30975 \times 1 \times 2 = 4.028.298.750$ Testpunkte für jedes Protokoll ergeben. Durch die in Abschnitt 6.4 diskutierte Optimierung kann die Testmenge auf die diagonalen Randpunkte reduziert werden, wodurch noch zwei aus acht möglichen Testpunkten $(2 \times 2 \times 1 \times 2)$ für jedes Protokoll zu

testen sind. Für den Test von TCP werden in dem Beispiel keine besonderen TCP-Flags gefordert, weswegen eine freie Wahl getroffen werden kann, die am besten der Protokollspezifikation entspricht. Sollten weitere Eigenschaften verlangt werden, so ist zu überprüfen, ob diese spezifikationskonform sind und den Paketfilter erreichen sowie von ihm verarbeitet werden können.

Aus den im Vektor beschriebenen Eigenschaften werden für die Tests konkrete Werte ausgewählt, die sich für die Überführung in ein Paket eignen. Als Beispiel werden hier die zwei Testpunkte

- 10.0.0.0, Port 1025 → 10.1.0.0, Port 80 und
- 10.0.0.255, Port 32000 → 10.1.0.255, Port 80

gewählt, die jeweils für beide Protokolle gelten sollen. Dabei wird angenommen, dass die gewählten Adressen aus Routingsicht keine besondere Bedeutung haben und zur Adressierung von Endsystemen verwendet werden können.

Ein Test für den Zustand *new*, also eine initiierende Assoziation, ist ohne weiteres möglich, indem für die zwei Testpunkte jeweils ein Paket versendet wird, das die gewählten Eigenschaften erfüllt. Komplizierter ist der Test für den established-Zustand, der eine „aufgebaute bidirektionale Assoziation“ erfordert. Hierfür muss der Proband vorbereitet und ein entsprechender Kontext hergestellt werden.

Der Kontext wird durch den Versand von Paketen hergestellt, die entsprechend der Protokollspezifikation für die Verbindungsverfolgung des Probanden erstellt werden. Die Untersuchung der Verbindungsverfolgung in Abschnitt 4.2 hat für den beispielhaft betrachteten netfilter-Paketfilter ergeben, dass bei TCP die TCP-Flags berücksichtigt und bei UDP lediglich ein „erstes“ und die „folgenden“ Pakete unterschieden werden. Dieser Kontext wird hier vorbereitet. Besonders zu berücksichtigen sind Fälle, in denen keine geeignete Vorgeschichte existiert oder die gewählte Vorgeschichte nicht hergestellt werden kann. Dann kann der Testpunkt nicht überprüft werden, was dem Benutzer mitgeteilt werden muss. Konnte die Vorgeschichte erfolgreich hergestellt werden, so wird ein geeigneter Test erstellt und ausgeführt.

Nach jedem versendeten Paket muss die Reaktion des Paketfilters analysiert und im Hinblick auf die erwartete Reaktion bewertet werden. Dabei können vier Reaktionsklassen unterschieden werden:

- Es kommt kein Paket an.
- Nur der Sender erhält ein Paket.
- Nur der Empfänger erhält ein Paket.
- Sender und Empfänger erhalten je ein Paket.

Abhängig von der erwarteten Reaktion und der Anzahl der erhaltenen Pakete müssen die Pakete weiter untersucht werden, wobei sowohl die erwartete Reaktion wie auch Fehlverhalten berücksichtigt werden. Für die Bewertung des Ergebnisses müssen die versendeten und die empfangenen Pakete auf eine Korrelation untereinander untersucht werden. Darunter ist zu verstehen, ob

- das empfangene Paket dem versendeten Paket entspricht und sich bei der Übertragung konfigurations- oder transferbedingte Felder verändert haben,
- ein falsches Paket aufgrund von Fehlverhalten oder -konfiguration empfangen wurde,
- das auf Senderseite empfangene Paket eine Fehlermeldung ist und sich gegebenenfalls auf das versendete Paket bezieht.

Dementsprechend werden auch mehrere Kategorien zur Bewertung des Endergebnisses eingeführt, die grob in OK, Warnung, Fehler und Empfangsfehler unterteilt und mit folgenden Bedeutungen belegt werden können:

OK	Das Ereignis stimmt mit der Erwartung überein.
Warnung	Das Ereignis stimmt mit der Erwartung fast überein. Eine Paketveränderung ist aufgetreten.
Fehler	Ein unerwartetes Ereignis.
Empfangsfehler	Das empfangene und das gesendete Paket stimmen nicht überein. Möglicherweise wurde ein fremdes Paket empfangen.

Die dafür benötigten Vergleichsfunktionen können in protokoll- bzw. übertragungsbedingte sowie in probandenabhängige Fälle eingeteilt werden. Die erste Gruppe kann generisch von der Teststrategie zur Verfügung gestellt werden. Letztere müssen für jede das Paket verändernde Funktion in der firewallabhängigen Beschreibung zur Verfügung gestellt werden.

Der generische Teil benötigt zur Auswertung Angaben über die Senderichtung des Paketes, das gesendete Paket, die empfangenen Pakete auf beiden Seiten und eine Beschreibung der postulierten Reaktion des Paketfilters. Darauf aufbauend kann ein Auswertungsalgorithmus wie in Abbildung 7.3 dargestellt vorgehen.

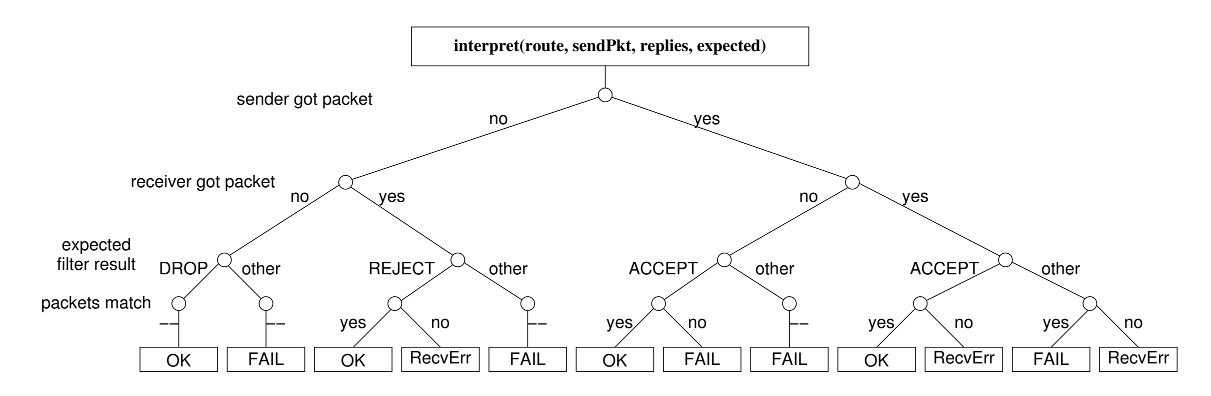


Abbildung 7.3: Entscheidungsgraph zur Testauswertung

Besonders wichtig bei der Entscheidung über das Endergebnis ist der letzte Vergleich, der überprüft, ob das empfangene Paket mit dem gesendeten Paket korreliert.

Dabei gibt es für die Ablehnung des Paketes bzw. der Assoziation durch den Paketfilter (*REJECT*) protokollabhängige Mechanismen zur Benachrichtigung des Senders. Allgemein sieht das Internet-Protokoll generische Mitteilungen über ICMP vor (z.B. *destination unreachable*). Für TCP kann aber auch ein TCP-Paket mit gesetztem RST-Flag verwendet werden. In jedem Fall muss überprüft werden, ob ein empfangenes Paket sich tatsächlich auf das gesendete Paket bezieht.

Im *ACCEPT*-Fall wird das empfangene Paket auf Veränderungen untersucht. Hier sind sowohl die vom Paketfilter beschriebenen Modifikationen (z.B. NAPT) wie auch die regulären und übertragungsbedingten Veränderungen (z.B. IP TTL) zu überprüfen. Die erwarteten Veränderungen werden von den Vektoren vorgegeben. Die vorgegebenen Aktionen werden auf firewallspezifische Vergleichsfunktionen abgebildet, die in Tabelle 7.3 ausschnittweise aufgeführt sind.

nat	ipt_DNAT, ipt_SNAT, ipt_MASQ, ...
mangle	ipt_TTL, ipt_ECN, ...
filter	ACCEPT, DROP, REJECT, ...

Tabelle 7.3: Beispiele und Kategorisierung der Auswertungsfunktionen

Nach der Auswertung aller Tests ergibt sich eine Hierarchie der Ergebnisse, wie sie in Tabelle 7.4 dargestellt sind. Aus jedem Vektor wird für jede Kombination aus Protokoll und Zustand ein Testfall abgeleitet (z.B. TCP-EST). Für jeden Testfall werden aus der association_list alle möglichen Pfade (*paths* zur Herstellung der Vorgeschichte berechnet. Jeder der Pfade wird auf Testbarkeit überprüft, wobei er verworfen wird, wenn

- in einem der Vektoren die geforderten Eigenschaften nicht spezifikationskonform sind oder die Übergänge keinen validen Protokollfluss etablieren können (in der Tabelle als *skip Protokoll* gekennzeichnet) oder
- die Vereinigung der definierten Adressen und Ports eine leere Menge ergibt, also keine verwendbaren Elemente übrig bleiben (*skip InterSection*).

Für jeden verwendbaren Testpfad werden daraufhin zwei Testpunkte (TP) bestimmt, die dann getestet werden können. Jeder Test der Testpunkte wird schließlich als erfolgreich oder fehlgeschlagen bewertet.

Diese Auswertung wird dem Benutzer in komprimierter Form dargestellt. Darauf basierend ist es möglich Aussagen über die Qualität des Regelwerks und der durchgeführten Tests zu treffen. Die fehlgeschlagenen Tests werden, sofern ermittelbar, in konfigurations-, spezifikations- und implementierungsbedingte Ergebnisse kategorisiert.

Übertragung des einfachen Tests auf den vollständigen Testraum

Nachdem der Testvorgang für einen Testvektor vereinfacht beschrieben wurde, wird jetzt das gleiche Verfahren auf den vollständigen Testraum übertragen. Dabei wer-

		paths	skip P	skip IS	test	TP	OK	error
Vektor 23, A2B								
TCP								
	INV	1	0	0	1	2	2	0
	NEW	1	0	0	1	2	2	0
	EST	3	0	1	2	4	4	0
	REL	3	3	0	0	0	0	0
UDP								
	...	...	...	...	...	...	...	...
Vektor 42, B2A								
TCP								
	...	...	...	...	...	...	...	...
Gesamt		768	323	97	348	696	651	45

Tabelle 7.4: Hierarchische Auswertung der Testergebnisse

den evtl. auftretende Probleme identifiziert, die im folgenden Abschnitt in eine neue Teststrategie umgesetzt werden.

Im Gegensatz zum einzelnen Vektor müssen nun die Vektoren für den vollständigen Testraum eingelesen und eventuell in ein internes Format der FWTStrategy gewandelt werden. Sofern möglich soll hier eine Prüfung der Konsistenz der Daten geschehen, bevor die aufwendigen Tests beginnen. FWTStrategy muss zusätzlich sicherstellen, dass alle benötigten Strukturen tatsächlich vorhanden sind und nicht z.B. eine ältere Version des Dateiformats oder gar eine unbekannte Datei eingelesen wurde.

Durch die vergrößerte Anzahl der Testvektoren muss nun eine Testreihenfolge bestimmt werden. Hier wird der einfachste Fall betrachtet, bei dem die Vektoren in der Eingabereihenfolge getestet werden.

Die Herstellung des Kontextes für einen Testpunkt im höheren Zustand setzt voraus, dass der Paketfluss der Vorgeschichte vom Paketfilter akzeptiert und weitergeleitet wird. Dabei wird von der Verbindungsverfolgung des Probanden ein Zustandseintrag für diesen Testpunkt vorgenommen, der für einen protokollabhängigen Zeitraum gespeichert wird. Die Testpunkte der Vorgeschichte sind aufgrund der Vollständigkeit des aufgespannten Testraums ebenfalls durch Vektoren beschrieben, was dazu führt, dass sie unabhängig von dem höheren Zustand zu einem anderen Zeitpunkt überprüft werden. Sollte diese Überprüfung innerhalb des Zeitfensters stattfinden, in dem die Verbindungsverfolgung noch einen Eintrag von dem früheren Test enthält, würde das Testergebnis verfälscht werden.

Bevor der Kontext für einen neuen Testpunkt hergestellt werden kann, muss also dafür gesorgt werden, dass sich die einzelnen Testfälle nicht gegenseitig beeinflussen. Hierzu könnten der Probandenbeschreibung die Timeouts für die Zustandsverfolgung entnommen und die folgenden Tests damit verzögert werden.

Analyse des Testablaufs

Bei der direkten Übertragung des einfachen Tests auf den vollständigen Testraum zeigen sich Schwächen des Testablaufs in Bezug auf die Testlaufzeit, die auf mehrere

Ursachen zurückgeführt werden können.

Die unabhängige Herstellung des Kontextes für jeden einzelnen Testpunkt führt dazu, dass Testpunkte mehrfach getestet werden, obwohl gerade die Reduzierung der notwendigen Testpunkte ein Ziel der Teststrategie ist. Weiterhin verursacht das Abwarten der Timeouts wiederholend Leerlauf. Das Abwarten wäre auch nur bei „kurzen" Timeouts unkritisch. Der PF-Paketfilter z.B. speichert laut Tabelle 4.2 die Zustandseinträge für eine aufgebaute TCP-Verbindung für 86400 Sekunden, bevor sie bei Leerlauf der Verbindung wieder gelöscht werden. Netfilter speichert laut Abbildung 4.10(b) die Einträge sogar für bis zu 432000 Sekunden. Hier muss eine neue Strategie eingeführt werden, um trotzdem zügig weiter testen zu können.

Die gewählte Abarbeitung in der Reihenfolge, in der die Vektoren in der Datei vorliegen, könnte die gegenseitige Beeinflussung der Testfälle weiter begünstigen. Eventuell lässt sich eine Reihenfolge finden, die die Laufzeit und die Beeinflussung reduziert.

Der bisherige Testablauf setzt die Anforderung der effizienten Ressourcennutzung also ungenügend um und sollte optimiert werden.

7.2. Entwicklung einer optimierten Teststrategie

In diesem Abschnitt werden die Kritikpunkte aus dem vorherigen Abschnitt aufgegriffen und in einer neuen optimierten Teststrategie für einen vollständigen Test umgesetzt.

In der Analyse wurden vier Ansätze zur Optimierung der Teststrategie aufgezeigt: die Reduzierung der Mehrfachtests und des Leerlaufs, Umgang mit „verbrannten" Testpunkten in der Zustandsverfolgung, Umstellung der Testreihenfolge und die Parallelisierung der Tests.

Entwicklung der Testreihenfolge

In der neuen Teststrategie werden die Testfälle im Kontext zueinander betrachtet und dafür verwendet, die Vorgeschichte für Testpunkte in höheren Zuständen herzustellen. Das kann bei der Anordnung der Testfälle ausgenutzt werden, indem die Tests vom höchsten zum niedrigsten Zustand abgearbeitet werden. Durch diese Anordnung wird erreicht, dass beim Aufbau der Vorgeschichte für die ersten Tests zum Teil bereits Tests aus den anderen Teilmengen benötigt werden und im Nachhinein nicht mehr zusätzlich getestet werden müssen, es sei denn sie werden wieder zur Herstellung einer Vorgeschichte benötigt. Die Definition der Zustände und deren Reihenfolge wird der Firewallbeschreibung entnommen. Danach wird die Teststeuerung, wie in Listing 7.1 dargestellt, mit dem zu testenden Protokoll und Zustand aufgerufen.

Ermittlung der Abhängigkeiten zwischen den Vektoren

Für die Bestimmung der Verzahnung und der Abhängigkeiten zwischen den Vektoren wird der FWA verwendet. Dieser liefert in der sogenannten *association list* eine

```
algorithm: teststeuerung
input: protokoll, zustand, vektorliste, Liste der Assoziationen (aList)
output: teststatistik

  testListe := finde Vektoren mit (protokoll, zustand) in vektorliste

  for vektor in testListe
    baum = generiere Baum: vektor, protokoll, Zustand, aList
    if baum is leer then
      melde "Keine Vorgeschichte zum Vektor"
      next vektor

    tpfad = neue Liste
    for pfad in baum
      for testschritt in pfad
        testfall := erstelle Testfall aus testschritt
        hänge testfall an tpfad an

      if not tpfad ist Testbar then
        melde "Pfad ist nicht testbar"
        next pfad
      vorlage = vereinige Vektoren im tpfad
      if not vorlage then
        melde "Pfad ist nach Vereinigung der Vektoren nicht testbar"
        next pfad
      pfad abbrechen

    testpunkte := suche Testpunkte nach vorlage
    for tpunkt in testpunkte
      übertrage tpunkt auf vorlage
      teste tpfad mit vorlage
      erstelle Testbericht
  return teststatistik
```

Listing 7.1: Algorithmus: teststeuerung. Hauptmethode der Teststeuerung

Vorauswahl von möglichen Vektoren, die sich zur Herstellung einer Vorgeschichte eignen. Diese Vorauswahl beruht auf der alternierenden relativen Senderichtung und der notwendigen Voraussetzung, dass der angegebene Kandidat den gleichen oder einen niedrigeren Zustand in seiner Definition miteinschließt. Die Überprüfung der Kandidaten mit der Protokollspezifikation wird aber von FWTStrategy übernommen.

Aus der Kandidatenliste in der *association list* wird eine Baumstruktur mit dem Zielvektor an der Wurzel erstellt. Jeder Pfad von der Wurzel bis zum letzten Knoten stellt einen möglichen Testpfad dar, der eine Kette vom höchsten zum niedrigsten Zustand aufbaut. Jeder dieser Testpfade muss auf Testbarkeit im Sinne der Protokollspezifikation überprüft werden, bevor er freigegeben wird. Andernfalls wird der nächste Pfad überprüft. Nach Möglichkeit wird dabei auch versucht, bereits verwendete Vektoren nicht mehrfach zu testen, indem ein alternativer Pfad gesucht wird.

Es gibt Fälle, in denen keine geeignete Vorgeschichte für den Zielvektor exis-

```
algorithm: generiere Baum
input: Wurzelvektor, Protokoll, Zustand, Liste der Assoziationen (aList)
output: Baum
  // Eintrag in aList: (Element) -> (Vorelement)

  knoten := neuer Knoten(target, proto, state)
  teilliste := finde Einträge für (Wurzel, Protokoll, Zustand) in aList
  for element in teilliste
    vorelement := ermittle Vorelement zu Element
    teilbaum := generiere Baum(vorelement, aList)
    hänge teilbaum an knoten
  return knoten
```

Listing 7.2: Algorithmus: generiere Baum. Generiert Baumstruktur aus der Liste der Assoziationen.

tiert oder hergeleitet werden kann. Dies kann passieren, wenn die Schnittmenge der möglichen IP-Adressbereiche und der Ports zwischen den ausgewählten Vektoren im Testpfad leer ist, was also abhängig vom Regelwerk ist. Die Algorithmen, die diese Überprüfung durchführen sind in Listing 7.3 und Listing 7.4 angegeben. Weiterhin können die in den Vektoren geforderten Eigenschaften nicht spezifikationskonform sein, so dass keine „funktionierenden“ Pakete daraus erstellt werden könnten. Dann kann der Testpfad nicht überprüft werden und der Benutzer wird entsprechend darüber in Kenntnis gesetzt.

Überprüfung des Testpfades

Bevor ein Pfad tatsächlich freigegeben wird und getestet werden kann, muss er auf Konformität zur Protokollspezifikation überprüft werden. Bei der Überprüfung muss einerseits jeder Vektor im Testpfad für sich dahingehend betrachtet werden, ob die beschriebenen Eigenschaften spezifiziert sind und andererseits, ob die Übergänge zwischen den Elementen im Testpfad hergestellt werden können. Hierzu werden für jedes Protokoll spezielle Prüffunktionen implementiert, die diese protokollabhängigen Eigenschaften kapseln.

Verwaltung der Testpunkte

Damit die Tests und die Pakete sich nicht gegenseitig beeinflussen, müssen die benutzten Tupel aus IP-Adresse und gegebenenfalls Port oder Typ für die sendende und empfangende Seite reserviert werden. Die Kombination von Quell-IP, Quell-Port, Ziel-IP und Ziel-Port sowie das Protokoll bilden einen Identifikationsschlüssel, der den Verbindungskontext für TCP und UDP eindeutig beschreibt. Das ICMP-Protokoll kann, sofern der eigenständige Teil aus Anfragen und Antworten betrachtet wird, eindeutig über Quell-IP, Ziel-IP sowie das Paar ICMP-Typ und -Code beschrieben werden. Der kontextabhängige Teil der ICMP-Fehlermeldungen muss nicht separat

```
algorithm: vereinige Vektoren
input: Liste von Vektoren
output: Schnittvektor or null

  schnittvektor := Vektor aus Vektorliste
  for vektor in Vektorliste
    if senderichtung(vektor) ≠ senderichtung(schnittvektor) then
      Richtung des vektors umdrehen
    schnittvektor.src := schneide(schnittvektor.src, vektor.src)
    schnittvektor.dst := schneide(schnittvektor.dst, vektor.dst)
    for feld in schnittvektor
      if Größe des Feldbereiches <= 0 then
        return null
  return schnittvektor
```

Listing 7.3: Algorithmus: vereinige Vektoren. Bildet die Schnittmenge der Eigenschaften mehrerer Vektoren.

```
algorithm: schneide
input: Bereich min_a,max_a, Bereich min_b,max_b
output: Bereich min_c,max_c

  if min_a <= min_b then
    min_c := min_b
    if max_a <= max_b then
      max_c := max_a
    else
      max_c := max_b
  else
    min_c := min_a
    if max_a <= max_b then
      max_c := max_a
    else
      max_c := max_b
  return (min_c, max_c)
```

Listing 7.4: Algorithmus: schneide. Bildet Schnittmenge von zwei Mengen.

behandelt werden, da er sich auf einen Kontext der bereits beschriebenen Tupel bezieht. Deswegen werden das erste Paket, das einen Kontext aufbaut, und die umgekehrte Richtung für die Antworten für eine protokoll- und zustandsabhängige Dauer gespeichert.

```
algorithm: suche Testpunkte
input: Testvektor
output: Tickets für reservierte Testpunkte

  dim := anzahl der eindeutigen Protokollmerkmale im Vektor
  for d in permutiere {min,max}^dim
    d' := inverses von d
    if d bereits benutzt or d' bereits benutzt then
      next
    if testpunkt d gesperrt or testpunkt d' gesperrt then
      stell (d,d') in Warteschlange
    else
      registriere Ticket für d und d'
  if not Tickets then
    warte auf Entsperrung von min(Warteschlange)
    registriere Ticket für d und d'

  return Tickets
```

Listing 7.5: Algorithmus: suche Testpunkte. Erstellt und reserviert Tickets für einen Testpunkt aus einem Vektor.

Für die Verwaltung dieser Tupel wird ein einfaches Ticketsystem verwendet[1]. Dort können Tickets für die Tupel angefordert und beim Versenden von einem Paket verlängert werden. Unabhängig davon, wie viele Pakete versendet werden, werden in einem Ticket automatisch die Hin- und die Rückrichtung reserviert.

Die Reservierungen werden an die Timeouts der jeweiligen Zustandsverfolgung geknüpft und erst nach Ablauf des Timeouts wieder frei gegeben. Für verbindungsorientierte Protokolle wie TCP kann der finale Timeout durch einen Verbindungsabbau verkürzt werden – bei anderen Protokollen muss jedoch abgewartet werden.

Zum Registrieren einer Verbindung und dem Ausstellen der Tickets müssen aus den im Vektor noch als Bereiche angegebenen IP-Adressen und Ports konkrete Werte aus den Randwerten der Schlüsseleigenschaften ausgewählt werden. Pro Testpfad werden so zwei Testpunkte bestimmt, für die auch zwei Tickets ausgestellt werden. Die Testpunkte müssen die diagonalen Eigenschaften, die in Abschnitt 6.4 beschrieben wurden, erfüllen. Tabelle 7.5 zeigt Beispiele für die Wahl der Testpunkte.

Sollte zum gegebenen Zeitpunkt keine Diagonale gebildet werden können, z.B. weil alle möglichen Testpunkte noch von früheren Tests reserviert sind, so wird aus den zurückgemeldeten Timeouts der kürzeste gewählt. Liegt dieser in einem konfi-

[1] Das Ticketsystem erfüllt keine Sicherheitsansprüche und verhindert es nicht, dass nur der ursprüngliche Empfänger des Tickets dieses verlängern darf.

Testpunkt 1 (src,dst,sport,dport)	Testpunkt 2 (src,dst,sport,dport)
`min, min, min, min`	`max, max, max, max`
`min, min, min, max`	`max, max, max, min`
`min, min, max, min`	`max, max, min, max`
`min, min, max, max`	`max, max, min, min`
`min, max, min, min`	`max, min, max, max`
`min, max, min, max`	`max, min, max, min`
`min, max, max, min`	`max, min, min, max`
`min, max, max, max`	`max, min, min, min`

Tabelle 7.5: Beispiel für die Wahl von Testpunkten. Die Testpunkte werden aus den Eigenschaften Quell-IP, Ziel-IP, Quellport und Zielport gebildet.

```
algorithm: entferne gesperrte Adressen
input: min und max eines Adressbereiches,
  Liste mit gesperrten Adressen
output: neuer Adressbereich

  if min in Liste mit gesperrten Adressen then
    inkrementiere min
  if max in Liste mit gesperrten Adressen then
    dekrementiere max
  if min verändert or max verändert then
    wiederhole enferne gesperrte Adressen mit neuem min,max
  else
    return min,max
```

Listing 7.6: Algorithmus: entferne gesperrte Adressen. Entfernt IPs von Produktivsystemen aus dem Testbereich.

gurierbaren Zeitfenster, so wird abgewartet – andernfalls wird der Testpfad in einer Warteschlange abgelegt und ein weiterer Testpfad untersucht. Anschließend wird die Warteschlange immer wieder konsultiert, um die dort abgelegten Tests abzuarbeiten.

Diese Phase wird ebenfalls dazu benutzt, den Betrieb in einer Realumgebung zu erlauben und Adressen von realen Systemen von den Testpunkten auszuschließen. Dazu werden die gesperrten Adressen in der Testkonfiguration beschrieben und mit den Testpunkten abgeglichen. Der Algorithmus in Listing 7.6 beschreibt die Vorgehensweise.

Ausführung des Tests

Der Ablauf der Testausführung ist in Listing 7.7 beschrieben. Aus dem erstellten Testpfad und den reservierten Adressen in den Tickets wird eine Vorlage mit den gemeinsamen Eigenschaften für die folgenden Testfälle erstellt. Die Vorlage kann auf jedes Element des Testpfads angewendet werden, woraufhin parametrisierte Testfälle

entstehen. Jeder Testfall wird aus einem Vektor abgeleitet und beschreibt dann eindeutige Eigenschaften eines Testpaketes. Diese werden an die Protokollmodule weitergereicht, die daraus Pakete generieren und verschicken können.

```
algorithm: testen
input: testvorlage, testpfad
output: Liste von Testergebnissen

  kontext := neuer TestKontext
  teile := trenne testpfad nach protokollen
  for teilpfad in teile
    pfad := neue Liste
    for testvektor in teilpfad:
      test' := erstelle testvorlage aus testvektor
      hänge test' an pfad an

    teste Pfad: pfad, protokoll, kontext
    if test_fehlgeschlagen then
      melde "Fehler - Herstellung der Vorgeschichte fehlgeschlagen"
      abbrechen
    else
      kontext := ergebnisse

  return ergebnisse
```

Listing 7.7: Algorithmus: testen. Steuert die Durchführung eines Testlaufs

Jedes Protokollmodul implementiert eine Funktion zur Überprüfung eines Teilpfades, welcher nur das jeweilige Protokoll enthält. Von dort aus werden aus jedem Testfall im Testpfad entsprechende Pakete erstellt, versendet und ausgewertet. Die Auftrennung des Pfades in mehrere Teilpfade wird für das Herstellen von Kontexten für unterschiedliche Protokolle (z.B. ICMP-Fehlermeldungen folgend auf TCP-Pakete) benötigt. Erst nachdem ein erwartungsgemäßes Ergebnis für ein Paket bestätigt werden konnte wird das nächste Paket oder der nächste Schritt im Testpfad ausgeführt.

Auswertung

Die Auswertung der Testfälle in der neuen Teststrategie unterscheidet sich konzeptionell nicht von der Auswertung für einen einzelnen Vektor. Bei der Auswertung der Vorgeschichte können aber die Ergebnisse für jeden Testschritt einem Vektor zugeordnet werden.

Die Informationen und Ergebnisse der Auswertung werden gesondert auf Testfall- und Paketebene zur weiteren Auswertung gesammelt, um daraus einen Bericht erstellen zu können. Bei einer abweichenden Reaktion der Firewall wird eine entsprechende Mitteilung an den Benutzer generiert und die folgenden Tests gegebenenfalls entsprechend neu bewertet bzw. angeordnet.

Erstellung der (Teil)Berichte

Bereits während der Ausführung des Tests werden dem Benutzer Teilergebnisse angezeigt, um den Fortschritt beobachten zu können. Die „Lautstärke" der Ausgaben ist über Kommandozeilenschalter konfigurierbar. In der Standardeinstellung werden nur unerwartete Ergebnisse detaillierter ausgegeben.

Am Ende aller Tests wird zusätzlich eine Übersicht erstellt, die die ausgeführten Tests nach Ergebnis aufschlüsselt. Die genaue Form und Bedeutung der Ausgaben wird im Benutzerhandbuch in Abschnitt A.3 beschrieben.

Offene Verbindungen behandeln

Beim Umgang mit „verbrannten" Testpunkten sind zwei Szenarien zu unterscheiden: die gegenseitige Beeinflussung von Tests innerhalb einer und zwischen mehreren Programmausführungen.

Nachdem der letzte Test gelaufen und ausgewertet ist, kann es passieren, dass aus den vorangegangenen Tests noch Kontextinformationen auf der Firewall vorhanden sind, die einen weiteren Testlauf beeinflussen könnten. Diese Informationen sind ebenfalls im Ticket-System gespeichert und können ausgewertet werden. Für verbindungsorientierte Protokolle wie TCP können diese Informationen dazu genutzt werden, die noch aktiven Verbindungen zu beenden. Für alle anderen Protokolle müssen diese Informationen persistent gespeichert werden, so dass sie beim Neustart von FWTStrategy eingelesen und gegebenenfalls berücksichtigt werden können. Dadurch wird die Wiederholbarkeit der Tests gewährleistet.

Die Parametrisierung der Testfälle muss zwischen den Testläufen identisch bleiben. Die Lösung der Anforderung ist durch das Anlegen einer Logdatei zum anschließenden Einlesen trivial zu lösen.

8. Proof of concept

Im vorherigen Kapitel wurde die Teststrategie konzeptionell erarbeitet. Diese Vorüberlegungen wurden in einem *proof of concept* prototypisch für den Paketfilter netfilter/iptables umgesetzt.

Durch den gesetzten Zeitrahmen für eine Diplomarbeit und die gewählten Werkzeuge konnte nur ein Teil der konzeptionell erarbeiteten Teststrategie umgesetzt werden.

Der Firewall-Analyser konnte zum Zeitpunkt dieser Arbeit die Konfiguration einer netfilter/iptables Firewall interpretieren und analysieren. Andere Konfigurationssprachen sollen in Zukunft hinzugefügt werden. Dadurch war nur eine Testumgebung verwendbar und die Anwendbarkeit der Teststrategie auf anderen Plattformen konnte nicht überprüft werden. Zur Entwicklung wurde der FWA in den Versionen der Serie 0.9.x verwendet, worauf sich die folgenden Beschreibungen beziehen. Getestet wurde mit den Standard netfilter-Komponenten im Linux Kernel 2.6.x bis 2.6.20 und iptables in der Version 1.3.6.

Die Tests konnten nur für den vermittelnden Aspekt der Konfigurationen umgesetzt werden, da die aktuellen FWTest-Agenten nicht auf einer filternden Netzwerkkomponente betrieben werden können. Deswegen konnten die von einer Firewall ausgehenden oder die dorthin gerichteten Assoziationen bei der Teststrategie nicht getestet werden. Diese Einschränkung reduzierte die Testabdeckung, kann aber prinzipiell nur für Firewallsysteme geändert werden, auf denen der Administrator auch Fremdsoftware nachinstallieren kann. Dies schließt die meisten Hardware-Appliances aus.

Bei der Auswahl der unterstützten Filteraktionen und Filterkriterien von iptables musste aus Zeitgründen eine Auswahl getroffen werden. Die Liste der ausgewählten Funktionen wird in Tabelle 8.1 wiedergegeben. Die wichtigsten Optionen, die das Verhalten des Paketfilters abweichend zu den verwendeten Standardwerten verändern können, sind der Tabelle A.2 zu entnehmen. Die Untersuchung dieser Optionen wird vom Prototyp nicht unterstützt.

In den folgenden Abschnitten werden die praktische Umsetzung, das Design und die konkreten Tests beschrieben. Im ersten Unterabschnitt werden die Aufteilung der Funktionalitäten auf die Module sowie die wichtigsten Funktionen und Algorithmen beschrieben. Der zweite Abschnitt stellt die Beschreibung des Probanden – in dem Fall netfilter/iptables – vor. Anschließend werden die funktionellen Aspekte der realisierten Protokolltests aufgezeigt. Im letzten Abschnitt wird ein Ausblick auf mögliche Erweiterungen und die Fortführung der Teststrategie gegeben.

Am Anfang des Kapitels 7 wurden Anforderungen an die Teststrategie definiert, die in diesem Abschnitt wieder aufgegriffen werden, um die Konzepte zu ihrer Umsetzung

Filterentscheidungen
IP: source, destination, TTL, TOS, DSCP, ECN
TCP: sport, dport
UDP: sport, dport
ICMP: type, code
state: INV, NEW, EST, REL
Filteraktionen
ACCEPT, DROP, REJECT
DNAT, SNAT, MASQUERADE

Tabelle 8.1: Liste der unterstützten iptables-Merkmale im Prototyp

zu erläutern.

Die Durchführbarkeit unter Realbedingungen impliziert die Unterscheidung zwischen Labor- und Produktivumgebungen, wobei letztere durch ein reales Netzwerk mit weiteren Systemen, zusätzlich zu der Testumgebung, gekennzeichnet sind. Um den Betrieb der sonstigen Systeme nicht zu beeinflussen, aber auch den Testablauf selbst gegen Störeinflüsse zu sichern, werden die Testagenten für jeden Testfall mit speziellen Eingangsfiltern konfiguriert, die den Empfang von Fremdpaketen ausschließen sollen. Gleichzeitig kann die Teststeuerung bei der Wahl der Testpunkte Adressen von Produktivsystemen oder spezielle Netzadressen auslassen. Die Teststrategie muss sich beim Versand der Pakete nicht um die Zuordnung der Adressen zu den Schnittstellen der Firewall kümmern, da der FWA relative Routing-Angaben liefert, die die Richtung des zu transferierenden Pakets aus der Sicht des Testers und der Agenten beschreiben. Das Testskript kann also direkt auf eine *A-Seite* und eine *B-Seite* der fwtest-Agenten referenzieren. Weiterhin kann der zu untersuchende Adressbereich über einen Schalter beeinflusst werden, indem der FWA bei der Transformation des Regelwerks angewiesen wird, statt den gesamten möglichen Bereich (0.0.0.0 bis 255.255.255.255) nur die tatsächlich verwendeten und adressierbaren Adressbereiche auszugeben.

Die Unabhängigkeit von konkreten Produkten und von den betrachteten Protokollen kann dank der vorgestellten Modellierung und einem modularisierten Aufbau der Teststrategie, bei dem die geräte- und protokollspezifischen Eigenschaften gekapselt werden, erreicht werden.

8.1. Modulkonzept

Die Vektoren werden vom FWA bereits als Pythonstrukturen ausgegeben, die in der Teststeuerung direkt eingelesen und interpretiert werden können. Das genaue Format dieser Datei ist im Anhang in Abschnitt A.2 dokumentiert.

FWTStrategy nutzt die Beschreibung des Paketflusses und der Firewallaktionen für die Zuordnung der symbolischen Aktionsbezeichner, die vom FWA übergeben werden, zu den firewallabhängigen Funktionen.

Dazu werden die Aktionen entsprechend der Tabelle 5.2 in die Klassen *NAT*, *modify*, *filter*, *inspect*, *route*, *handle_fragment* und *handle_state* eingeteilt, für die jeweils

spezialisierte Funktionsmodule implementiert werden. Sie sind einerseits für die Abbildung der Vektoren in Paketeigenschaften, aber auch für die anschließende Auswertung der empfangenen Pakete verantwortlich. Die Definition der Aktionen kann dafür verwendet werden, die Berechnungen für terminierende Pfade vorzeitig zu beenden, um Ressourcen zu sparen.

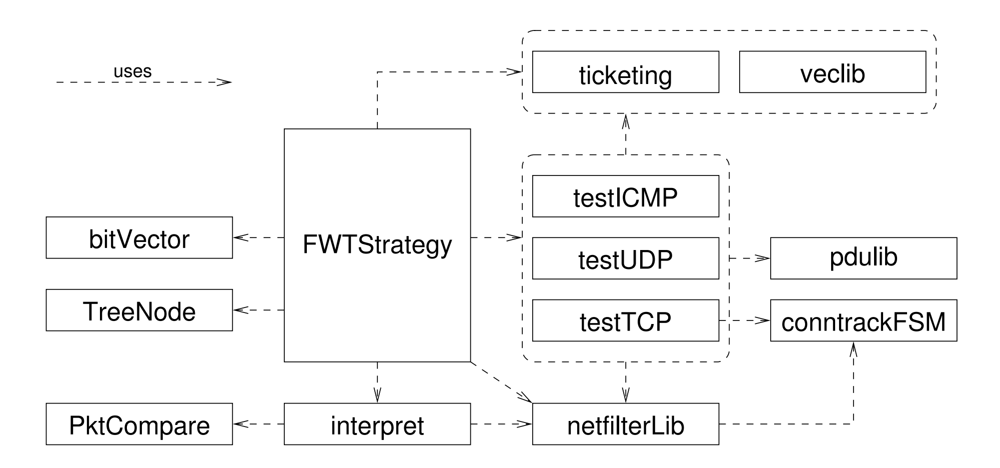


Abbildung 8.1: Modularchitektur der Teststrategie

Der grobe Ablauf der Teststrategie wurde bereits in Abschnitt 7.2 vorgestellt. In Ergänzung dazu wird hier die Interaktion der einzelnen Module beschrieben.

Abbildung 8.1 zeigt die Modularchitektur der Teststrategie. Zentrale Komponente der Teststategie ist die Teststeuerung FWTStrategy. Sie steuert und interagiert mit verschiedenen Modulen, die für verschiedene Aufgaben zuständig sind und Dienste bzw. Schnittstellen für die anderen Module anbieten.

Alle Funktionen zum Einlesen und Verarbeiten der Vektoren sind in der Funktionsbibliothek `veclib` zusammengefasst. Darunter sind auch Funktionen zum Überführen der Vektoren in die intern verwendeten Testfälle.

Die firewallspezifischen Definitionen, Funktionen und Strukturen sind für den im Prototyp berücksichtigten Paketfilter in der `netfilterLib` enthalten. Zu den wichtigsten Komponenten dieses Moduls gehören die Beschreibung der TCP- Zustandsverfolgung und die Funktionen zur Auswertung der Aktionen bzw. zur Abbildung der Filterkriterien in den Paketeigenschaften. Die Zustandsverfolgung setzt auf einem generischen Automaten aus dem Modul `conntrackFSM` auf.

Zur Berechnung und Verwaltung der Testfälle sowie der Testpunkte werden die Module `bitVector`, `TreeNode` und `ticketing` verwendet. Mit Hilfe der Struktur `bitVector` und den darauf basierenden Operationen ist es möglich besonders einfach alle diagonalen Testpunkte für einen gegebenen Testfall zu bestimmen. In `TreeNode` wird die Basisstruktur zum Aufbau eines Baumes aus der Association List und der Traversierung aller Pfade implementiert. Im Modul `ticketing` wird das bereits eingeführte Ticketsystem implementiert, das die reservierten, die aktiven und die „verbrannten“ Testpunkte verwaltet.

Die spezifischen Tests der Protokolle, die auf den jeweiligen Protokollspezifikatio-

nen basieren, sind in eigenen Modulen umgesetzt. Im Prototyp sind die Protokolle TCP, UDP und ICMP berücksichtigt. Sie alle implementieren jeweils die Logik für die Herstellung des jeweiligen Protokollflusses und Methoden zur Bewertung der Testpfade. Die Funktionen zum Versenden und Empfangen der Pakete, auf die diese Module zugreifen können, sind in der `pdulib` gekapselt. Das Modul `testTCP` greift zusätzlich auf die Paketfilter-spezifische Zustandsverfolgung zu.

Die letzten beiden Module im Schaubild dienen der Auswertung der Testergebnisse. Das Modul `interpret` vergleicht das gesendete Paket mit dem bzw. den empfangenen Paketen und der postulierten Aktion. Zum Vergleich der Pakete nutzt es die Funktionen aus dem `PktCompare`-Modul.

Die Verwaltung der Testpunkte durch das Ticketsystem, der Basisautomat der Zustandsverfolgung, die Umsetzung der Protokollspezifikationen und die Probandenbeschreibung sind essentielle Bestandteile der Teststrategie, die in den folgenden Unterabschnitten beschrieben werden. Auf die weitere Beschreibung der Auswertung wird verzichtet, da die zugrunde liegende Vorgehensweise bereits in dem Abschnitt zur Abbildung 7.3 beschrieben ist. Die restlichen Module werden nur als Hilfsfunktionen betrachtet und deswegen nicht weiter beschrieben.

8.1.1. Ticketsystem

Das Ticketsystem besteht aus den vier Klassen ***`ticketing`***, ***`eventlog`***, ***`logItem`*** und ***`timedRingBuffer`***. Ersteres ist die Hauptklasse, die in der Teststrategie verwendet wird und die anderen dienen zur Verwaltung der Tickets. Die Klassen sind in Abbildung 8.2 dargestellt.

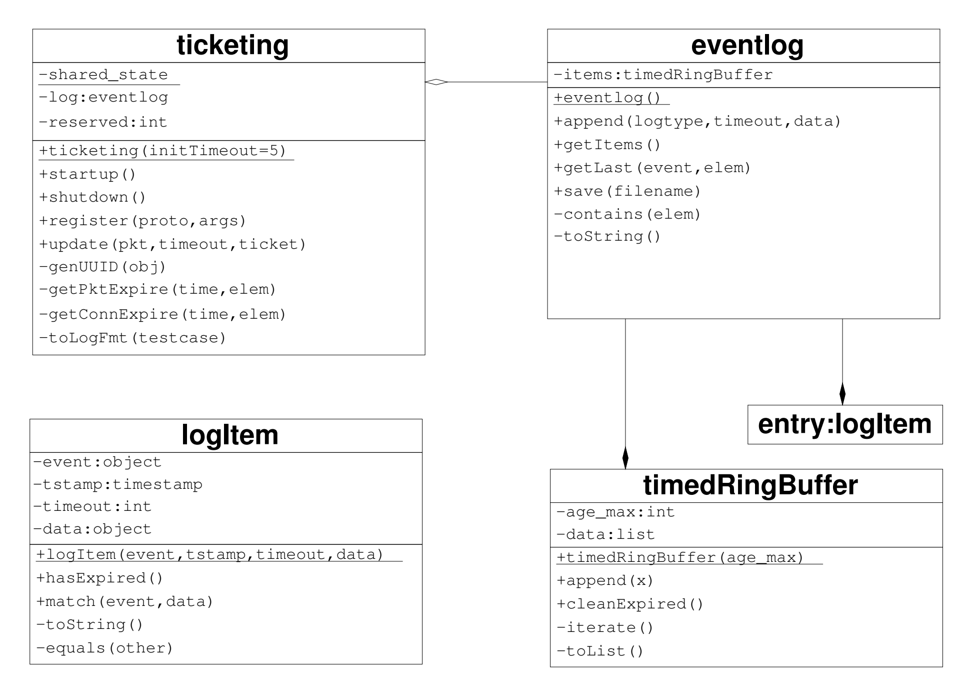


Abbildung 8.2: Klassen des Ticketsystems

class ticketing

shared state – Die Klasse ticketing ist nach dem *Monostate Pattern* modelliert, damit alle Protokollmodule auf ein gemeinsames Ticketsystem zugreifen können. In der Klassenvariablen *shared state* wird eine Referenz auf den gemeinsamen Zustand gespeichert.

ticketing() – Konstruktorfunktion. Kann optional mit dem initialen Timeout für neue Tickets parametrisiert werden.

startup() – Lädt gespeicherten Zustand aus einer Datei, um die Tickets zu berücksichtigen und Kollisionen bzw. Einflüsse aus früheren Aufrufen zu verhindern.

shutdown() – Speichert den aktuellen Zustand in einer Datei.

getStatus(proto,args) – Überprüft evtl. ein Ticket, das für das Protokoll *proto* und die Eigenschaften *args* ausgestellt wurde. Falls ein noch gültiger Eintrag existiert wird eine verbleibende Gültigkeitsdauer, die größer Null ist, zurückgeliefert.

register(proto,args) – Fordert ein initiales Ticket, ausgestellt für das Protokoll *proto* und die Eigenschaften *args*, an. Die Anzahl und Art der Argumente ist variabel. Falls bereits ein Ticket, das für die gleichen Parameter ausgestellt wurde, noch gültig ist, wird aktiv abgewartet. Andernfalls wird ein neues Ticket mit der voreingestellten Gültigkeitsdauer ausgestellt.

update(pkt,timeout,ticket) – Aktualisiert ein ausgestelltes Ticket mit dem Paket und setzt einen neuen Timeout.

genUUID(obj) – Private Funktion. Erstellt einen eindeutigen Identifier für ein als Parameter übergebenes Objekt. Wird als Teil des Tickets verwendet.

getPktExpire(time,elem) – Private Funktion. Sucht ein Ticket, das für das in *elem* beschriebene Element ausgestellt wurde und liefert die verbleibende Gültigkeitsdauer.

getConnExpire(time,elem) – Private Funktion. Liefert die verbleibende Gültigkeit in Sekunden des Elements *elem* zum Zeitpunkt *time*.

class eventlog

Das *eventlog* speichert die Tickets, die von dem Ticketsystem ausgestellt werden.

eventlog() – Konstruktorfunktion. Hier wird ein Behälter für die Logeinträge als *timedRingBuffer* angelegt.

append(logtype,timeout,data) – Fügt dem eventlog einen neuen Eintrag vom Typ *logtype*, einer Gültigkeit für die Dauer von *timeout* und dem Inhalt *data* hinzu. Der Typ wird für die Unterscheidung der Ereignisse Registrierung (lock),

Paket gesendet (send), Paket empfangen (received) und die Abfrage (any) verwendet.

getItems() – Liefert eine Liste aller Logbucheinträge zurück.

getLast(event,elem) – Liefert den neuesten Eintrag für das Ereignis *event, elem*. *event* kann ein konkreter Ereignistyp oder der Abfragetyp any sein.

save(filename) – Speichert den Inhalt des Logs in die Datei *filename*.

contains(elem) – Private Funktion zur Feststellung, ob ein Eintrag für das Element *elem* bereits vorhanden ist.

class logItem

Die Einträge des Logbuchs werden in Objekten vom Typ *logItem* abgelegt.

logItem(event,tstamp,timeout,data) – Konstruktorfunktion. Erstellt einen Logbucheintrag, der über das Ereignis *event*, den Zeitstempel *tstamp*, eine Gültigkeitsdauer *timeout* und den Inhalt *data* beschrieben wird.

hasExpired() – Abfragefunktion zur Feststellung, ob der Logeintrag entsprechend des Zeitstempels und der Gültigkeitsdauer noch gültig ist.

getTimestamp() – Liefert den Zeitpunkt der Erstellung des Eintrags zurück.

match(event,data) – Vergleichsfunktion. Liefert einen Wahrheitswert, ob dieser Eintrag mit einem über *event* und *data* beschriebenen Eintrag übereinstimmt.

class timedRingBuffer

Der *timedRingBuffer* ist ein Ringspeicher, der nur Einträge bis zu einer voreingestellten Dauer behält und sie danach verwirft.

timedRingBuffer(age_max) – Konstruktorfunktion. Legt einen *timedRingBuffer* mit der maximalen Gültigkeitsdauer von *age_max* für dessen Einträge.

append(x) – Fügt der vorliegenden Instanz einen neuen Eintrag *x* hinzu.

cleanExpired() – Entfernt alle Einträge aus dem Speicher, die die voreingestellte Gültigkeitsdauer überschritten haben.

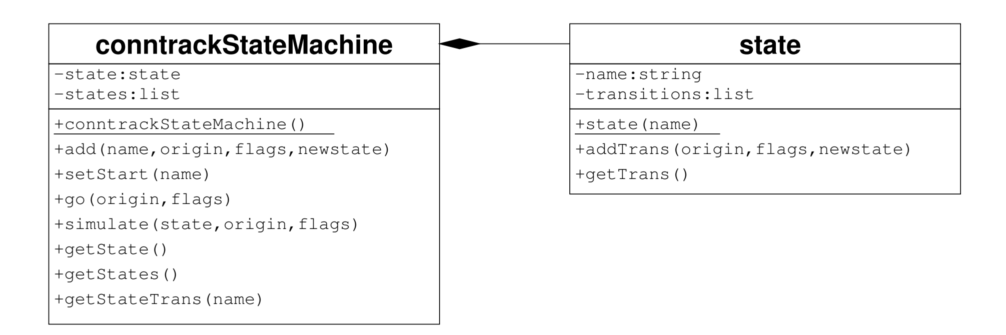


Abbildung 8.3: Klassen der TCP Zustandsverfolgung

8.1.2. Automat zur Beschreibung der Zustandsverfolgung

class conntrackStateMachine

Die hier modellierte conntrackStateMachine entspricht einem Mealy-Automaten, da die Ausgaben bei Zustandsübergängen (die Timeouts) entsprechend der Beschreibung zur Abbildung 4.10 abhängig von dem Zustand und der Eingabe sind. Die Signalisierung eines validen und nicht ignorierten Übergangs wird über die Ausgabe des neuen Zustands realisiert. Für ignorierte Eingaben wird der reservierte Zustandsbezeichner `IGNORE` verwendet.

conntrackStateMachine() – Konstruktorfunktion. Liefert eine neue Instanz der *conntrackStateMachine*.

add(name,origin,flags,newstate) – Fügt der Instanzvariablen *states* eine neue Transition mit der Bezeichnung *name*, einer Eingabe bestehend aus Sender *origin* und Flags *flags* sowie dem Folgezustand *newstate* hinzu. Die vorgegebenen Senderdefinitionen beinhalten Server, Client und Beliebig.

setStart(name) – Definiert den Zustand *name* als Startzustand.

go(origin,flags) – Versucht mit der Eingabe *origin* und *flags* eine Transition durchzuführen. Bei einer validen Eingabe wird der neue Zustandsname oder `IGNORE` zurück geliefert und der neue Zustand in der Instanzvariablen *state* vermerkt.

simulate(state,origin,flags) – Simuliert den Übergang ausgehend von dem Zustand *state* mit der Eingabe von *origin* und *flags* und liefert ein Ergebnis analog zu *go()* ohne den neuen Zustand intern zu speichern.

getState() – Liefert den aktuellen Zustand zurück.

getStates() – Liefert eine Liste aller definierten Zustände zurück.

getStateTrans(name) – Liefert die definierten Transititionen ausgehend von dem Zustand *name* zurück.

8.1.3. Beschreibung des Probanden

Obwohl der Firewall-Analyzer aktuell nur eine Konfigurationssprache unterstützt, setzt FWTStrategy laut Aufgabenstellung eine allgemeine Teststrategie, die sich an die Merkmale einer konkreten Firewall anpassen und konfigurieren lässt, um. Die eigentlichen Ausgaben vom FWA sind selbstbeschreibend und abstrakt gehalten, so dass auch andere Firewalls in Zukunft damit beschrieben werden können. Beim Einlesen der Daten und der Einstellung der Teststrategie auf das jeweilige Modell muss den Bezeichnern jedoch eine Semantik zugeordnet werden. Diese produktspezifischen Eigenschaften müssen deswegen entsprechend beschrieben werden.

Unter den benötigten Informationen wurden die Gruppen Verbindungsverfolgung, NAT-Einstellungen, Routing und die Zuordnung von Ports zu durchgeführten Anwendungsinspektionen identifiziert. Letztere sind notwendig, um bei tieferen Untersuchungen des Paketfilters (*deep inspection*) zumindest korrekte Musterpakete auf Anwendungsebene erstellen oder bei Misserfolg den Benutzer ausführlicher informieren zu können. Im Prototyp wird bei TCP-Verbindungen maximal ein vollständiger TCP-Handshake ausgeführt, bei dem keine Daten ausgetauscht werden dürfen. Deswegen wird eine Beschreibung der Ports und Daten nicht benötigt. Für UDP wurden beispielhaft Nutzlasten für einfache Request-Reply-basierte Protokolle wie DNS oder NTP hinterlegt, die bei Bedarf von der Strategie verwendet werden. ICMP trägt in der Regel keine Nutzlast oder bezieht sich auf einen vorhandenen Kontext, so dass auch hier keine eigene Nutzlast benötigt wird.

Für die gezielte Herstellung von bestimmten Zustandseinträgen sowie zum Zwecke der Bestimmung von Timeouts werden die Anzahl, die Art sowie die Übergänge zwischen den Zuständen benötigt. Unterschieden wird dabei zwischen den Zuständen der jeweiligen Protokollzustandsmaschinen (z.B. TCP oder UDP/DNS) und den Zuständen der abstrakteren Verbindungsverfolgung (z.B. NEW, ESTABLISHED, RELATED,...). Die Abhängigkeiten zwischen den Zuständen der Verbindungsverfolgung werden indirekt durch die association list vom FWA vorgegeben. Lediglich die Reihenfolge der Abarbeitung wird in der Probandenbeschreibung hinterlegt. Bei iptables wird die Reihenfolge

1. ICMP-REL, ICMP-EST, ICMP-NEW, ICMP-INV
2. TCP-REL, TCP-EST, TCP-NEW, TCP-INV
3. UDP-REL, UDP-EST, UDP-NEW, UDP-INV

verwendet. Die Beschreibung der TCP-Zustandsmaschine ist mittels der vorgestellten *`conntrackStateMachine`* realisiert und mit den in Abbildung 4.10 beschriebenen Übergängen parametrisiert.

Für die Interpretation der SNAT-Variante Masquarading/Hide-NAT sind Angaben zu den konfigurierten externen Adressen der Firewall bzw. zu den sogenannten NAT-Adresspools und der konfigurierten Portbereiche notwendig. Sollte eine Firewall eine getrennte Verbindungsverfolgung für NAT verwenden, so müssten auch

hier die Zustände mit ihren Übergängen und den Timeouts spezifiziert werden. Dieses Szenario wird aber von FWTStrategy aktuell nicht unterstützt. Alle anderen NAT-Angaben sind dem Regelwerk direkt zu entnehmen und können getestet werden. Die Angaben zu den Adressbereichen und den Firewalladressen werden aktuell in der FWTest-eigenen Konfiguration hinterlegt und von der Teststrategie verwendet.

8.1.4. Protokollmodule

Jedes Protokollmodul enthält Funktionalitäten und „Wissen“, die für das Verbindungsmanagement des jeweiligen Protokolls erforderlich sind. Im Allgemeinen lösen die Module folgende Aufgaben:

- Prüfen der Korrektheit des Testpfades
- Anpassen des Paketformates zu dem getesteten Verbindungszustand
- Pakettransfer zwischen den Agenten
- Rückgabe der Auswertung des Tests
- Speichern der Statistiken über alle gesendeten Paketen.

Die Protokollmodule testTCP, testUDP und testICMP enthalten Funktionalitäten, die für die folgenden Aufgaben zuständig sind:

- Pakettransfer zwischen zwei Agenten
- Prüfen der Korrektheit des Testpfades. Zuvorstehende Testfälle müssen eine spezifikationskonforme Vorgeschichte für den Nachfolger herstellen
- Alle gesendete Pakete zählen. Am Ende des Tests können die Statistiken der Firewall mit den Zählern der Strategie verglichen werden. Damit kann der ordnungsgemäße Ablauf des Tests gestützt werden.

Jedes Modul hat eine nach außen hin definierte Schnittstelle und verwendet intern auch vergleichbare Hilfsfunktionen bzw. Strukturierung.

isTestable(testcase) – Diese Funktion wird von FWTStrategy mit einem ***testcase*** als Parameter aufgerufen, um festzustellen, ob sich aus den dort beschriebenen Eigenschaften ein testbares Paket ableiten lässt. Hier werden erste Überprüfungen der Konformität zur Spezifikation durchgeführt.

checkTestpath(testpath) – Diese Funktion wird von FWTStrategy mit dem Parameter ***testpath***, also der Zusammenfassung mehrerer Testfälle, aufgerufen, um festzustellen, ob dieser Testpfad einen testbaren Protokollfluss darstellt. Die Übergänge zwischen den Paketen im Protokollfluss werden überprüft.

testPath(testpath, context) – Dies ist die eigentliche Funktion zur Herstellung des in *testpath* beschriebenen Protokollflusses unter Berücksichtigung des Kontextes aus *context*. *testPath()* liefert eine Liste der Ergebnisinterpretation für alle durchgeführten Tests zurück. Dafür benutzt es die von *interpret()* zur Verfügung gestellte Schnittstelle.

sendPacket(testcase, ...) – Private Funktion zur Herstellung und zum Transfer der Pakete, die aus dem *testcase* sowie gegebenenfalls einer weiteren protokollspezifischen Parametrisierung abgeleitet werden.

report() – Funktion zum Abrufen einer Statistik über die gesendeten Pakete in Abhängigkeit von der geforderten Phase der Assoziation. Wird am Ende aller Testläufe von FWTStrategy benutzt, um für den Benutzer einen Bericht zu erstellen.

8.2. Umsetzung der Protokolltests

Implementierung der UDP-Tests

Der Prototyp der FWTStrategy ist in der Lage UDP-Datenflüsse für die von Netfilter unterschiedenen Zustände INVALID, NEW und ESTABLISHED zu erzeugen.

Der RELATED-Zustand wurde nicht implementiert, da hierfür ein Anwendungsprotokoll implementiert werden müsste, für das Netfilter eine Verbindungsverfolgung anbietet. In der betrachteten Version wären das die UDP-basierenden Protokolle TFTP, SIP und Amanda[1].

INVALID: In der Dokumentation zu Netfilter sind keine genauen Kriterien zum UDP-INVALID-Zustand gegeben, die jedoch dem Quellcode[2] entnommen werden konnten. Dort werden Pakete mit einer falschen Prüfsumme im UDP-Header sowie unvollständige bzw. zu kurze Pakete als INVALID-Paket eingestuft.

NEW: Ein valides UDP-Paket, das zu keiner aktiven Assoziation gehört. Das erste Paket in einer Assoziation, das Netfilter sieht, wird als NEW-Paket interpretiert.

ESTABLISHED: Ein Paket, das zu einer der Verbindungsverfolgung bekannten Assoziation gehört. Das bedeutet, dass vor dem Testen eines ESTABLISHED-Paketes eine entsprechende NEW- oder ESTABLISHED-Assoziation aufgebaut werden muss.

[1] Der Advanced Maryland Automatic Network Disc Archiver (Amanda) ist ein Backup-System. Siehe `http://www.amanda.org` für weitere Informationen.

[2] Analysiert wurde die Datei `linux/net/ipv4/netfilter/ip_conntrack_proto_udp.c`

Implementierung der TCP-Tests

Beim Verwalten einer TCP-Verbindung gibt es eine größere Menge von Kriterien als bei den anderen Protokollen, die berücksichtigt werden müssen. Außer IP-Adressen und TCP-Ports müssen unterschiedliche Flag-Kombinationen sowie die Übergänge und Zustände in der Zustandsmaschine der TCP-Zustandsverfolgung (siehe Abbildung 4.10) für das Erzeugen der TCP-Pakete berücksichtigt werden.

Die prototypische Implementierung ist nicht in der Lage TCP-Verbindungen im RELATED-Zustand zu erzeugen, da hierfür wie bei UDP die Implementierung von Anwendungsprotokollen notwendig wäre. Netfilter unterstützt die Verbindungsverfolgung von den (teilweise) auf TCP basierenden Protokollen H323, FTP, PPTP, IRC und NetBIOS.

INVALID: Wie bei UDP wird der INVALID-Zustand bei TCP mit einem Paket mit einer falschen Prüfsumme erzeugt. Die Verbindungsverfolgung ordnet auch bestimmte Flag-Kombinationen dem INVALID-Zustand zu.

NEW: Der NEW-Zustand wird durch ein Paket erzeugt, das keiner Assoziation in der Verbindungsverfolgung zugeordnet werden kann. Es ist möglich den NEW-Zustand mit zahlreichen Flag-Kombinationen zu erreichen, nicht jede davon lässt sich jedoch aufgrund der internen Zustandsmaschine in den ESTABLISHED-Zustand bringen. Die Strategie beschränkt sich darauf das SYN-Flag zu setzen.

ESTABLISHED: Voraussetzung für das Auslösen einer ESTABLISHED-Regel ist ein vorhandener Eintrag in der Verbindungsverfolgung. Das Paket muss mit dem gleichen Adress-Port-Tupel sowie einer geeigneten Flag-Kombination verschickt werden.

Nicht jedes Paket, das als ESTABLISHED interpretiert wird, ändert den internen Zustand der TCP-Statemachine.

Implementierung der ICMP-Tests

Eine ICMP-Assoziation wird im Netfilter eindeutig durch IP-Adressen, ICMP-Type, ICMP-Code und Identifier definiert. Der Identifier wird nur von bestimmten Typen verwendet.

Jeder Zustand der ICMP-Assoziation kann nur mit Paketen mit bestimmten ICMP-Typen erreicht werden. In der Implementierung wurden ICMP-Typen aus [RFC792; RFC950; RFC1108; RFC2002] betrachtet. Alle ICMP-Typen, die beim Testen berücksichtigt werden, sind in fwt_constants.py definiert.

Nicht alle ICMP-Verbindungen lassen sich in der aktuell verwendeten Testumgebung testen. Deshalb wurden einige aus den Tests ausgeschlossen:

ICMP type 5, Redirect message – Hierbei handelt es sich um eine Aufforderung des Gateways an den Sender zur direkten Adressierung des Zielrechners. Das

Gateway stellt dabei fest, dass Sender und Empfänger sich im gleichen Adressierungssegment befinden und eine Vermittlung nicht notwendig ist. Nachrichten von diesem Typ werden vom lokalen IP-Stack der Firewall spezifikationskonform verworfen, falls sie eine Netzsegmentgrenze überschreiten sollten (vgl. Abschnitt 3.2.2.2 in [RFC1122]).

ICMP type 9/10, Router Advertisement/Selection message – mit diesen ICMP-Typen wird Kommunikation zwischen Routern realisiert, mit der sie sich gegenseitig durch Advertisement/Selection-Nachrichten suchen können. Die Konstruktion einer funktionierenden Router Advertisement Nachricht ist nicht trivial und benötigt ein komplexeres Szenario (vgl. [RFC1256]). In der Entwicklungsumgebung war es nicht möglich diesen Fall nachzubilden, wobei die Pakete bei den Versuchen von der Firewall verworfen wurden.

NEW: Der NEW-Zustand kann nur mit ICMP-Paketen vom Typ *request message* aufgebaut werden. Dazu gehören:

- ICMP type 8, Echo request message
- ICMP type 13, Timestamp request message
- ICMP type 15, Information request message
- ICMP type 17, Address mask request message

ESTABLISHED: Der ESTABLISHED-Zustand in der ICMP-Assoziation wird durch das Senden einer Antwort auf eine ICMP Anfrage erzeugt, die zu der Anfrage passt. Zu den Antworttypen gehören folgende Typen:

- ICMP type 0, Echo reply message
- ICMP type 14, Timestamp reply message
- ICMP type 16, Information reply message
- ICMP type 18, Address mask reply message

Für eine korrekte Zuordnung der Pakete in der Verbindungsverfolgung muss der Identifier im Antwortpaket mit der Identifikationsnummer der Anfrage übereinstimmen. Andernfalls kann Netfilter das Paket keinem bekannten Datenstrom zuordnen und wertet es als INVALID.

ICMP-Anfragen werden in der Verbindungsverfolgung gezählt, so dass die Anzahl der Antworten mit der Anzahl der Anfragen übereinstimmen muss. Das bedeutet, dass eine Anfrage den Zähler inkrementiert und ein erwartetes Antwortpaket den Zähler dekrementiert. Stehen noch mehrere Antworten aus, so kann ein ESTABLISHED-Eintrag in der Verbindungsverfolgung beobachtet werden. Erreicht der Zähler die Nullposition wird der Verbindungseintrag gelöscht. Der Timeout des ESTABLISHED- Zustands für ICMP beträgt 0 Sekunden, was bei gleichverteilten Anfragen und Antworten dazu führt, dass der ESTABLISHED-Zustand nicht beobachtet werden kann.

RELATED: Eine der Hauptaufgaben von ICMP ist die Fehlerbenachrichtigung und Kontrolle der Verbindungen. Dadurch werden ICMP-Pakete in bestimmten Fällen als RELATED, also einem anderen Datenstrom zugehörend, interpretiert. Netfilter betrachtet alle ICMP vom Typ *error message* als RELATED. Dazu gehören:

- ICMP type 3, Destination unreachable message
- ICMP type 4, Source quench message
- ICMP type 5, Redirect message
- ICMP type 11, Time exceeded message
- ICMP type 12, Parameter problem message.

Zusätzlich müssen sie in der Nutzlast den IP-Header und mindestens 8 Byte der höheren Protokollschicht von einem Paket enthalten, das bereits registriert wurde und der Verbindungsverfolgung bekannt ist.

Um eine ICMP-Assoziation im RELATED-Zustand aufzubauen muss zuerst ein geeignetes Paket gesendet werden, das die Konfiguration passieren lässt und von der Firewall weitergeleitet wird. Daraufhin kann in Gegenrichtung ein ICMP-Paket von den genannten Typen mit dem zuvor gesendeten Paket (zumindest der IP-Header und 8 Byte der höheren Schicht) in der Nutzlast gesendet werden.

INVALID: Die INVALID-Assoziation wird durch Senden eines Antwortpaketes erzeugt, wenn keine entsprechende Anfrage gesendet wurde. Alternativ ist es auch möglich eine ICMP Fehlermeldung zu senden, die sich auf keine bekannte Assoziation bezieht.

8.3. Ausblick auf künftige Erweiterungen

In diesem Abschnitt wird ein Ausblick auf die weiteren Entwicklungsmöglichkeiten der FWTStrategy gegeben. Es soll als eine technische Zusammenfassung der aktuellen Strategie und mögliche Erweiterungen vorstellen.

Als problematisch bei dem gewählten Ansatz hat sich das Einlesen der Vektordatei herausgestellt. Die ganze Datei wird auf einmal über die Pythonfunktion `execfile()` evaluiert, wobei alle erstellten Objekte gleichzeitig im Speicher gehalten werden. Dies führt bereits bei 1000 Listen mit je 1000 Elementen in der Vektordatei zu einem Speicherverbrauch von etwa 380MB und steigt fast linear im Verhältnis zu der Anzahl der Listen an. Diese Größenordnung der Listen ist nicht nur theoretischer Natur und kann bereits durch relativ einfache Firewallkonfigurationen entstehen. Bei den Untersuchungen hat sich weiter herausgestellt, dass Python die Objekte anders verwaltet, wenn die gleiche Menge an Listen in einer Schleife erzeugt wird. Aus diesem Grund wird eine eigene Implementierung der Evaluierungsfunktion für eine mögliche Fortführung des Projekts empfohlen.

Eine alternative Form der Teststrategie könnte dazu genutzt werden bei der Rekonstruktion einer unbekannten Firewall und ihrer Einstellungen zu helfen. Methoden zur Ermittlung der verwendeten TCP-Zustandsmaschine und den voreingestellten Timeouts können untersucht werden.

Eine weitere Anwendungsform der Teststrategie könnte die Reaktion der Firewall auf verschiedene bekannte Angriffsformen untersuchen. Ähnliche Werkzeuge existieren bereits, sie führen jedoch Black-Box-Tests durch. Hier könnte der Einfluss der Konfiguration im Grey-Box-Verfahren betrachtet werden.

9. Ergebnisse

In diesem Kapitel werden mögliche Anwendungsfälle der entworfenen Teststrategie und die erzielten Ergebnisse aus durchgeführten Testläufen vorgestellt. Die Anwendungsfälle sollen den praktischen Nutzen der Arbeit belegen. In Abschnitt 9.2 wird die Testabdeckung durch den Prototyp für ausgewählte Konfigurationen und Verbesserungspotenzial durch Fortentwicklung der Teststrategie aufgezeigt. Der anschließende Abschnitt 9.3 fasst die Erkenntnisse aus der Entwicklung der Teststrategie und der Untersuchung des netfilter-Paketfilters zusammen.

9.1. Anwendungsfälle für FWTStrategy

Konzept und Design der entwickelten Teststrategie unterstützen das Testen der Funktionalität und der Konformität von vermittelnden Knoten mit dem Schwerpunkt Paketfilter. Besonders interessant ist es damit verschiedene Konfigurationen von gleichen oder unterschiedlichen Produkten zu vergleichen, aber auch ein Abgleich des Verhaltens mit der Spezifikation ist möglich.

Funktionale Tests

Im einfachsten Anwendungsfall werden der Paketfilter und die Teststrategie mit der gleichen Konfiguration eingestellt. Damit ist es möglich für jeden vom FWA gelieferten und testbaren Vektor eine Aussage über den Vergleich der erwarteten und des beobachteten Ereignisses zu treffen.

Die Einschränkung der Aussage auf testbare Vektoren schließt folgende Fälle aus:

1. Der Vektor beschreibt Eigenschaften, aus denen sich kein spezifikationskonformes Paket ableiten lässt. Ein solches Paket kann nicht verschickt werden.
2. Ein spezifikationskonformes Paket kann abgeleitet werden, aber aus den vorhandenen Vektoren lässt sich der notwendige Kontext nicht herstellen.
3. Ein spezifikationskonformer Paketfluss kann abgeleitet werden, aber die Implementierung der Teststrategie ist nicht vollständig, so dass die geforderten Eigenschaften nicht umgesetzt werden können. Eine Erweiterung der Teststrategie ist notwendig.

Die ersten beiden Fälle entstehen durch die vollständige Berechnung des Testraums im FWA, wodurch nicht nur Positiv-Vektoren, sondern auch die Negative entstehen. Unter Positiv-Vektoren werden diejenigen verstanden, die sich aus expliziten

Angaben der Konfiguration ergeben. Die Negativ-Vektoren entstehen durch die Vervollständigung der expliziten Angaben auf den gesamten Testraum. Die nicht spezifikationskonformen und damit nicht testbaren Vektoren werden von der Teststrategie identifiziert und mit einer entsprechenden Begründung gemeldet. Sie können der Testabdeckung zugerechnet werden, da sie im Betrieb nicht auftreten können oder von anderen Routern bzw. dem Protokollstapel verworfen werden.

Der dritte Fall kann durch die Eingabe einer Konfiguration entstehen, die durch die Teststrategie nicht unterstützte Filterekriterien bzw. -entscheidungen enthält. Dann kann zu den betroffenen Vektoren keine Aussage getroffen werden, wodurch Lücken in der Testabdeckung entstehen können. Die von der prototypischen Implementierung unterstützten Eigenschaften und Funktionen sind in Tabelle 8.1 aufgeführt.

Nach einem Testlauf von FWTStrategy stellt sich die Frage nach der Aussagekraft der gelieferten Ergebnisse. Hierbei können folgende Fälle unterschieden werden:

- Teststrategie und Paketfilter arbeiten ordnungsgemäß,
- Fehlerfall in
  - den Testvektoren,
  - dem Modell bzw. der Beschreibung des Paketfilters,
  - der Teststrategie,
  - dem Paketfilter,
  - mehr als einem Teil.

Wenn alle Teile ordnungsgemäß arbeiten und die Teststrategie alle geforderten Eigenschaften testen konnte, wird mit dem Existenzquantor nachgewiesen, dass in den Randbereichen aller identifizierten Segmente die postulierte Aktion des Paketfilters beobachtet werden konnte. Dabei wird wie in Abschnitt 6.4 beschrieben angenommen, dass der Paketfilter sich innerhalb der Segmente stetig verhält. Aus dem Ergebnis lässt sich nicht schließen, dass der Paketfilter die zugrunde liegenden Protokollspezifikationen vollständig und konform umsetzt.

Werden am Ende einer Testdurchführung Abweichungen zwischen den erwarteten und den beobachteten Ergebnissen festgestellt, so muss ein Fehler in den Testvektoren, dem Modell, dem Paketfilter oder der Teststrategie vorliegen.

Eine fehlerhafte Berechnung der Vektoren durch den FWA kann zu Abweichungen führen. Die Identifizierung und Beseitigung eines solchen Fehlers liegt in der Verantwortung des FWA-Entwicklers.

Das Modell wird bei der Zusammenstellung des Testpfades verwendet, um die spezifischen Eigenschaften der zustandsbehafteten Verbindungsverfolgung zu berücksichtigen. Ein Fehler im Modell oder der Beschreibung des Paketfilters müsste gleichartige Symptome an mehreren kontextabhängigen Testpunkten aufzeigen.

Die Teststrategie enthält zwei potenzielle Fehlerquellen: die Testdurchführung und die Testauswertung. Bei der ersteren können die erstellten Pakete von einer Spezifikation abweichen, die bei der Implementierung nicht (genügend) berücksichtigt wurde.

Die Abweichungen können durch einen Abgleich zwischen der Implementierung under Spezifikation gefunden werden. Der Einsatz eines Netzwerksniffers zum Aufzeichnen der gesendeten Pakete kann dabei hilfreich sein. Es kann aber auch vorkommen, dass der Test spezifikationskonform und erwartungsgemäß durchgeführt wurde, aber die Testauswertung das Ergebnis nicht korrekt zuordnen konnte. Dieser Fehlerfall kann durch den Vergleich der Vorgabe, des protokollierten Ergebnisses und der Fehlermeldung identifiziert werden. Weiterhin wird in der Testauswertung versucht, die Fehlerart zu klassifizieren.

Sollten die behandelten Maßnahmen nicht zur Lösung beigetragen haben, so erhärtet sich der Verdacht eines Fehlers im Paketfilter. Hierzu ist die Dokumentation bzw. die Spezifikation des Paketfilters oder der Hersteller zu konsultieren.

Abweichungen in mehreren Komponenten können, müssen aber nicht, sichtbar werden. Fehler können sich überdecken oder summieren.

Zur weiteren Entlastung eines der Beteiligten kann der abweichende Teil der Testkonfiguration mit einer anderen Produktversion oder einem anderen Produkt (manuell) wiederholt und verglichen werden. Die differenzierenden Testkonfigurationen werden im Folgenden beschrieben.

Vergleich von zwei Konfigurationen

Gegeben seien zwei unterschiedlich formulierte Konfigurationen für einen Paketfilter und die Behauptung, dass die (beobachtbare) Wirkung beider Konfigurationen gleich sei. Dann ist es mit FWTStrategy möglich, diese Behauptung zu überprüfen.

Dazu sind folgende Konfigurationen zu testen:

1. FWTStrategy mit Config A und Paketfilter mit Konfiguration A
2. FWTStrategy mit Config A und Paketfilter mit Konfiguration B
3. FWTStrategy mit Config B und Paketfilter mit Konfiguration A und
4. FWTStrategy mit Config B und Paketfilter mit Konfiguration B.

Werden bei den Testkonfigurationen Abweichungen zwischen den erwarteten und den beobachteten Ergebnissen festgestellt, so ist die Behauptung falsifiziert. Eine Verfifizierung ist aufgrund der unvollständigen Testabdeckung nicht möglich. Aus den Meldungen von FWTStrategy ist es aber möglich, einen Teil der Unterschiede zu identifizieren.

Vergleich von zwei Paketfilterversionen

Gegeben sei eine Konfiguration für einen Paketfilter. Diese wird auf zwei verschiedenen Versionen des Paketfilters installiert. FWTStrategy soll die Wirkungsunterschiede beider Paketfilter vergleichen.

Dazu sind folgende Konfigurationen zu testen:

1. FWTStrategy mit Config A und Paketfilter 1 mit Konfiguration A

2. FWTStrategy mit Config A und Paketfilter 2 mit Konfiguration A.

Werden zwischen beiden Testkonfigurationen Abweichungen festgestellt, so weisen beide Paketfilter Wirkungsunterschiede auf, bei denen sie sich nicht gleich verhalten. Der Umkehrschluss gilt nicht. Werden keine Abweichungen festgestellt, so ist es kein Nachweis, dass die Wirkung beider Paketfilter im gesamten Testraum gleich ist.

Sollte der FWA in Zukunft weitere Konfigurationssprachen unterstützen, so wäre es möglich die Strategie zu erweitern und auch Vergleiche zwischen verschiedenen Paketfiltern durchzuführen.

9.2. Beispiele der Testabdeckung

Die Qualität der entwickelten Teststrategie kann anhand der erreichten Testabdeckung gemessen werden. Auch wenn die Teststrategie nur prototypisch umgesetzt wurde, können bereits Ansätze und Tendenzen identifiziert werden. Im Folgendem werden Auswertungen der Testabdeckung von vier Konfigurationen vorgestellt.

Für die Auswertung wurden zwei reale Konfigurationen und zwei Testkonfigurationen ausgewählt. Die Konfiguration Uniseb2 gehört zu der ersten Kategorie und wird für die Anbindung eines Arbeitsraumes im Fachbereich KBS der Fakultät IV an das Uni-interne Netz verwendet. Gleichzeitig ist das auch die größte untersuchte Konfiguration. Rusty-2 ist eine reale Minimalkonfiguration zur Verbindung eines privaten Netzwerks mit einem Öffentlichen unter Verwendung von Masquerading. Die letzten beiden, clean und ICMP, sind einfache Konfigurationen, die zum Testen der Strategie erstellt wurden. Die Konfigurationen sind zustandsbehaftet und reglementieren die drei Protokolle TCP, UDP und ICMP. Uniseb2 und Rusty-2 verwenden neben ACCEPT, DROP und REJECT auch MASQUERADING als Aktionen.

Config	Vec#	TC Total	not implemented UTR	U-REL	T-REL	Bad Protocol Properties	NO Prehistory	In Test	Skip	Paths Skip	In Test	TP
Uniseb2	368	488	70	24	72	78	94	135	15	1224	257	514
ICMP	20	150	30	2	2	46	32	38	0	1032	38	76
Rusty-2	5	30	6	2	2	–	–	17	3	5	19	38
clean	2	30	6	2	2	–	–	14	6	2	22	44

Tabelle 9.1: Auswertung der Testabdeckung für Beispielkonfigurationen

Die Ergebnisse der Auswertung nach der Durchführung der Tests mit den Konfigurationen werden angelehnt an die Tabelle 7.4 in Tabelle 9.1 aufgeschlüsselt. Die Spalten sind wie folgt zu lesen:

Config – Name der Konfiguration.

Vektoranzahl – Anzahl der vom FWA berechneten Vektoren.

TC Total – Die Anzahl der Testfälle, die aus den Vektoren abgeleitet wurden. Sie sind die obere Schranke der möglichen Tests.

not implemented – Im Prototyp konnte nur eine Auswahl von Tests umgesetzt werden. Zu den fehlenden Tests gehört eine Umsetzung des untracked-Zustandes (UTR) der Verbindungsverfolgung sowie der related-Zustand für UDP und TCP. Letztere beiden benötigen die Implementierung eines Anwendungsprotokolls wie FTP oder H323, für die iptables spezielle Erweiterungen anbietet. Die Vervollständigung der Testabdeckung für den untracked-Zustand konnte in der gegebenen Bearbeitungszeit nicht gelöst werden. Alle drei Fälle lassen sich aber durch die Implementierung entsprechender Erweiterungen der Teststrategie ebenfalls abdecken.

Bad Protocol Properties – Diese Testfälle konnten wegen den im Vektor geforderten Protokolleigenschaften nicht umgesetzt werden. Diese Eigenschaften sind eine direkte oder indirekte Folge der Konfiguration und ergeben sich entweder durch die explizite Forderung durch eine Regel oder aus der Vervollständigung des Testraumes durch den FWA. Die Anzahl der nicht getesteten Testfälle für die explizit geforderten Eigenschaften lässt sich durch eine Konfigurationsänderung verändern.

No Prehistory – Für diese Testfälle konnte kein geeigneter Kontext gefunden werden, mit dem der Testfall vorbereitet werden könnte. Diese Gruppe fasst zwei weitere Fälle zusammen. Der FWA liefert in der *association list* keine Vorgeschichte zu diesem Testfall, weil keine Regel in der Konfiguration einen vorangegangenen Verbindungszustand erlaubt. Weiterhin ist es möglich, dass akzeptierende Regeln für die Verbindungszustände existieren, aber diese aufgrund der geforderten Protokolleigenschaften sich für keinen spezifikationskonformen Protokollfluss verwenden lassen. In beiden Fällen kann auch im regulären Betrieb des Paketfilters ein solcher Kontext nicht hergestellt werden.

In Test und Skip – Die unter *In Test* genannte Zahl bezeichnet die tatsächlich getesteten Testfälle, für die ein Protokollfluss hergestellt wurde. Die unter *Skip* zusammengefassten Testfälle wurde ebenfalls getestet, indem sie für die Herstellung eines Kontextes für andere Testfälle benutzt wurden. Sie werden aufgrund der eingebauten Optimierung der Laufzeit nicht nochmal separat getestet.

Paths: Skip und In Test – Die hier genannten Zahlen beschreiben die Anzahl der betrachteten Testpfade für die zu testenden Testfälle aus der vorangegangenen Rubrik. Dabei bezeichnet *Skip* die Anzahl der Testpfade, die aufgrund einer Verletzung der jeweiligen Protokollspezifikation nicht in einem Protokollfluss umgesetzt werden können. Die Zahl unter *In Test* sind die tatsächlich getesteten Pfade. Beide Zahlen sind unabhängig von der Anzahl der Vektoren oder der Testfälle. Die Anzahl der Testpfade hängt von der Konfiguration ab.

TP – Die letzte Spalte gibt die Anzahl der Testpunkte wieder. Sie lässt sich direkt aus der Anzahl der getesteten Testpfade ableiten, da für jeden Testpfad zwei diagonale Testpunkte ausgewählt werden (vgl. Abschnitt 6.4).

Die Testabdeckung kann als die aktive Überprüfung der geforderten Eigenschaften durch die Herstellung eines Protokollflusses oder als eine begründete Stellungnahme zum Auslassen des Tests verstanden werden. Ziel ist es möglichst vollständig die Eigenschaften aktiv zu überprüfen. Durch die Bindung aller Protokollflüsse an die Protokollspezifikationen können jedoch nicht alle Testfälle, die der FWA beschreibt, überprüft werden, weil die Vorbedingungen nicht erfüllt werden können.

	untracked	UDP related	TCP related	zusammen	gesamt
uniseb2	14.3%	4.9%	14.8%	34%	65%
ICMP	20%	1.3%	1.3%	22.6%	47.6%
rusty-2	20%	6.7%	6.7%	33.4%	90.4%
clean	20%	6.7%	6.7%	33.4%	80.4%

Tabelle 9.2: Potenzial zur Verbesserung der aktiven Testabdeckung

Das Maß der aktiven Testabdeckung ergibt sich aus dem Quotienten der getesteten Testfälle und deren Gesamtanzahl. Dabei beinahlten die getesteten Testfälle die aufgeführten Zahlen unter *In Test* und *Skip*. Darauf basierend ergeben sich für uniseb2 eine Testabdeckung von 31%, für ICMP 25%, für clean 47% und für Rusty-2 57%. Diese Testabdeckung kann bei der Ergänzung der Teststrategie durch die fehlenden Tests zwischen 22% und 34% gesteigert werden. Die letzte Spalte der Tabelle zeigt die dann erreichbare Gesamtabdeckung der Testfälle auf Basis der Testvektoren.

9.3. Erkenntnisse aus der Untersuchung von iptables

Bei der Entwicklung der Teststrategie und verschiedener Testszenarien konnten Abweichungen zwischen den erwarteten und den tatsächlichen Ergebnissen beobachtet werden. Zunächst wurde angenommen, dass es sich bei den Abweichungen um Fehler in der Implementierung oder undokumentiertes Verhalten handelt. Bei der weiteren Nachforschung zu den Abweichungen konnte jedoch das beobachtete Verhalten auf die im [RFC1812] beschriebenen Anforderungen an IPv4 Router zurückgeführt werden. Alle entdeckten Unstetigkeiten, die in Tabelle 9.3 zusammengefasst sind, sind in einer Spezifikation zu finden. Daraus folgt aber nicht, dass in netfilter/iptables keine Fehler bzw. Unstetigkeiten zu finden, oder dass die Anforderungen und Spezifikationen vollständig umgesetzt sind.

Protokoll	Eigenschaft	Policy	Reaktion	Begründung
all	src=0.0.0.0	REJECT	DROP	RFC1812, Sektion 5.3
all	src=255.255.255.255	REJECT	DROP	RFC1812, Sektion 5.3
ICMP	redirect message to fw [5]	all	DROP	RFC1122, Sektion 3.2
ICMP	error message [3,4,11,12]	REJECT	DROP	RFC1812, Sektion 4.3
ICMP	reply message [0,14,16,18]	SNAT	unverändert	INVALID, kein Kontext
TCP	flags=PUFARS (XMas)	SNAT	unverändert	INVALID, kein Kontext

Tabelle 9.3: Auflistung der entdeckten Abweichungen im netfilter-Verhalten

Die Analyse verschiedener Versionsstände des iptables Paketfilters zeigte Unterschiede auf, die sich dazu nutzen lassen spezielle Testkonfigurationen zu erstellen, mit denen sich die Versionen unterscheiden lassen. Damit wäre ähnlich einem Fingerabdruck eine Identifikation des Paketfilters in einem vermittelnden Knoten möglich.

Folgende Abweichungen wurden zwischen mehreren netfilter Versionen festgestellt:

Unterschiedliche Bedeutung von –syn im iptables frontend: iptables interpretiert die Angabe der Konfigurationsoption –syn zur Überprüfung der TCP-Flags bis zur Version 1.3.8 als „nur SYN aus FIN,SYN,RST,ACK gesetzt“. Mit der neuen Version wurde das FIN-Flag entfernt.

Fehler bei der Auswertung des RST-Flags in Linux 2.6.10: folgt das RST-Flag in der genannten Linux-Version direkt auf ein Paket mit gesetztem ACK-Flag, wird es ignoriert. Dies hat zur Folge, dass der Timeout nicht verkürzt wird und der Eintrag in der Verbindungsverfolgung weiterhin Gültigkeit behält.

Veränderungen der als gültig deklarierten TCP-Flags: die von der Verbindungsverfolgung akzeptierten und ausgewerteten TCP-Flags wurden mehrfach ergänzt. Die Kombination SYN,ACK,PSH wird seit dem 25.04.2005 akzeptiert, SYN,PSH seit dem 12.11.2005 und SYN,URG seit dem 05.03.2007.

Geänderter Timeout für die TCP-Verbindungsverfolgung: bei Linux Versionen vor 2.6.1 war der Timeout für den Zustand close_wait in der TCP-Verbindungsverfolgung auf drei Tage eingestellt. Am 3.12.2003 wurde dieser Timeout auf 60 Sekunden verkürzt.

Änderungen der Verbindungsverfolgung für TCP: ein Zustandsübergang scheint immer wieder neu interpretiert worden zu sein, so dass dieser mehrfach geändert wurde. Die folgende Tabelle fasst die Änderungen in einer Übersicht zusammen.

Datum	Version	Zustand	TCP-Flag	Reaktion
(heute)	<= 2.6.22	SynSent	server:ack	IGNORE
01.12.05	< 2.6.14.4	SynSent	server:ack	INVALID (DROP)
10.03.05	< 2.6.11.3	SynSent	server:ack	IGNORE
		SynRecv	server:ack	SynRecv (vorher INVALID)
14.12.04	< 2.6.10	SynSent	server:ack	INVALID (DROP)

Tabelle 9.4: Änderungen der Übergänge in der TCP-Zustandsmaschine

Die in der Tabelle und in der vorliegenden Arbeit beschriebene Zustandsmaschine wurde erst in Linux 2.6.9 eingeführt. Vorher wurden die TCP-Flags an mehreren Stellen im Code einzeln überprüft – ähnlich, wie es IPFilter, PF und IPFW bis heute noch machen.

10. Bewertung und Vergleich

In diesem Kapitel werden die entwickelte Teststrategie und das Werkzeug FWTStrategy in den Kontext anderer Arbeiten gestellt. Dabei wird die erarbeitete Lösung verglichen und bewertet.

In Abschnitt 10.1 werden zunächst Werkzeuge vorgestellt, die verschiedene Aspekte der Netzwerk- und Firewalltests umsetzen. Sie werden in zwei große Gruppen unterteilt und kurz vorgestellt.

Abschnitt 10.2 stellt verwandte wissenschaftliche Arbeiten vor, die sich mit dem Testen von Firewalls beschäftigen. Auch hier wird eine Kategorisierung eingeführt, um die Schwerpunkte der Arbeiten besser voneinander abgrenzen zu können.

In Abschnitt 10.3 werden schließlich die ähnlichsten Ansätze aus den Werkzeugen und den verwandten Arbeiten ausgewählt und mit der vorliegenden Arbeit verglichen. Am Ende des Vergleichs steht eine Bewertung der Teststrategie auf qualitativer und quantitativer Ebene.

10.1. Testwerkzeuge

Die hier vorgestellten Werkzeuge stellen nur eine kleine Auswahl der vorhandenen Software dar, die Netzwerk- und Sicherheitsanalysen durchführen können. Vor allem im kommerziellen Sektor gibt es Programme, die sich diesen Zielen widmen, aber dieser Untersuchung aus zeitlichen und finanziellen Gründen verschlossen bleiben[1].

Netzwerkanalyse, Penetration und Schwachstellenabtastung

Die einfachste Art ein Netzwerk oder eine Firewall zu untersuchen ist es die beobachteten Pakete passiv zu protokollieren (engl. sniffing) und zu analysieren. Das bekannteste Werkzeug dafür ist **Wireshark** [Wireshark] (ehemals ethereal). Die Netzwerkanalyse kann durch gezielte Injektion (engl. packet forging) von Paketen ergänzt werden. Dafür können **Scapy** [Scapy] oder **hping3** [Hping] verwendet werden. Beide können Pakete erstellen, senden sowie empfangen und verfügen jeweils über eine Skriptingschnittstelle. Scapy unterstützt dabei mehr Protokolle als hping3.

Hping wie auch Scapy und Wireshark sind für viele verbreitete Plattformen als Open Source verfügbar.

Eine verbreitete Methode die Sicherheit von Netzwerken zu testen sind die so genannten *penetration tests*, also der Versuch durch die Firewall durchzubrechen und

[1] Weitere Listen von relevanten Werkzeugen können z.B. unter `http://dmoz.org/Computers/Security/Internet/Products_and_Tools/Security_Scanners/` und `http://sectools.org/` abgerufen werden.

in ein verwundbares System einzudringen. Dazu gehört zunächst die Erkundung des Netzwerks auf potentielle Ziele und Eindringungspunkte. Zur ersten Gattung der unterstützenden Werkzeuge gehören Scanner und Netzwerkkartographen. Das bekannteste unter ihnen dürfte **nmap** [Nmap] sein. Es kann durch **amap** [Amap] unterstützt werden, das sich auf die Erkennung von Dienstanwendungen spezialisiert hat, selbst wenn sie auf einem unüblichen Port betrieben werden. Beide Anwendungen sind für viele Umgebungen als freie Open Source Programme verfügbar. **firewalk** [Firewalk] kann die vom Paketfilter weitergeleiteten Ports herausfinden. Die Funktionsweise wurde in [GS98] beschrieben und publiziert.

Firewalltest und -analyse

Der **AlgoSec Firewall Analyzer (AFA)** [AFA] ist ein kommerzielles Werkzeug, das das Regelwerk vieler Firewalls analysieren sowie potentielle Konflikte zwischen den Regeln und sicherheitskritische Einstellungen finden kann. Dies wird durch vollständige offline-Analyse mit einem auf der Firewall ANalysis enGine, kurz FANG, basierenden Simulationsmodell, das auf Seite 112 bei den verwandten Arbeiten vorgestellt wird, erreicht. Bei der offline-Analyse findet keine Interaktion mit der Firewall statt und das Regelwerk wird in einer Simulation analysiert. Dies hat einerseits den Vorteil, dass der Betrieb der Firewall nicht gestört wird, anderseits aber auch den Nachteil, dass die Analyse nur so genau ist, wie das Simulationsmodell es erlaubt und abweichende Verhaltensweisen des Systems nicht gefunden werden können. Neben der reinen Analyse bietet der AFA auch weitere Managementfunktionen wie Änderungsverfolgung, automatische Überprüfungen oder Regelwerkoptimierungen an.

An der Universität DePaul, Chicago/USA, ist im Rahmen mehrerer Arbeiten rund um die Erkennung und Behandlung von Anomalien in Firewall Policies der **Firewall Policy Advisor (FPA)** [FPA; ASH03] entstanden. Der FPA kann in einer vereinfachten Notation des Regelwerks, die das Protokoll, IP-Adressen, TCP- bzw. UDP-Ports und die Akion berücksichtigt, Anomalien auffinden und Korrekturvorschläge machen. Leider müssen aber die echten Konfigurationen zuerst manuell in die vereinfachte Notation gebracht und anschließend beide gepflegt werden.

Ebenfalls im Rahmen einer Forschungsarbeit ist an der University of Victoria, Canada, unter dem Namen **Blowtorch** ein C++ Framework für die Automatisierung von Firewalltests entstanden. Das Grundkonzept der Teststeuerung beruht auf einem Paketiterator, der ereignis- und zeitgesteuert Paketströme erzeugen kann. Die Bibliothek enthält zahlreiche Funktionen zur Erstellung, Analyse, Manipulation, Entsendung und zum Empfang von Paketen sowie zum (De-)Multiplexing von Datenströmen. Blowtorch ist nicht öffentlich verfügbar, wurde laut [HY05] aber erfolgreich für die Durchführung von Tests an verschiedenen Firewalls benutzt.

Der **Firewall Tester** bzw. FTester[2] von Andrea Barisani [FTester] ist ein Werk-

[2]Ein Artikel zur Einführung in die Benutzung des FTesters kann unter `http://www.tisc-insight.com/newsletters/56.html` abgerufen werden.

zeug zum Testen von Firewall-Regelwerken und Eigenschaften von Intrusion Detection Systemen (IDS). Es ist in der Skriptsprache Perl realisiert und besteht aus einem Paketinjektor sowie einem Paketempfänger. Die Testdaten werden über eine Konfigurationsdatei definiert und vom Paketinjektor versendet. Beide Seiten legen jeweils Log-Dateien an, aus deren direktem Vergleich die geblockten oder veränderten Pakete ermittelt werden können. Der Testprozess und die dazugehörige Konfiguration müssen manuell für jeden neuen Fall erstellt, analysiert und interpretiert werden. Leider bietet die Konfigurationsdatei keine Möglichkeit die Pakete genau zu spezifizieren und es lassen sich damit auch keine bedingten bzw. komplexeren Abläufe definieren.

Im Rahmen mehrerer Semesterarbeiten wurde an der ETH Zürich das Werkzeug **fwtest (ETH Zürich)** [fwtest] entwickelt. Mit Hilfe einer einfachen Konfigurationssprache können statische Testfälle definiert und sequentiell oder im begrenzten Rahmen auch parallel abgearbeitet werden. Ein Testfall besteht aus mehreren Paketen, für die jeweils das Protokoll, ausgewählte Protokollfelder, ein symbolischer Zeitschlitz sowie das erwartete Ergebnis angegeben werden können. Die gesamten Ergebnisse werden in einer Logdatei protokolliert. Auch dieses Tool ist nur eingeschränkt für komplexere Einsatzszenarien geeignet, da keine bedingten Abläufe möglich sind, die Abweichungen zwischen den gesendeten und den empfangenen Paketen nur grob definiert werden werden können und aktive Antworten der Firewall auch nicht vorgesehen sind.

10.2. Verwandte Untersuchungen

Zu Beginn der Vorbereitungen für diese Arbeit wurden zahlreiche verwandte Arbeiten gesucht und analysiert. Diese Nachforschungen dienten der besseren Einarbeitung in das Thema und bieten die Möglichkeit auf aktuellen Ergebnissen aufzubauen. Die Arbeiten werden, angelehnt an Abbildung 6.4 auf Seite 64, verschiedenen Abstraktionsebenen beim Testen von Firewalls zugeordnet und in diesen Kategorien vorgestellt.

Firewallsoftware – Produkttests

PBit ist eine Testmethodologie, die auf Vorlagen basierend parametrisierte Testfälle für Regressionstests des Paketfilters netfilter/iptables erstellen kann. Einige Vorlagen, Algorithmen, die diese kombinieren können sowie eine grafische Oberfläche wurden mitentwickelt [vgl. DH04].

„Testing iptables“ [HPS03] dagegen untersucht den Zusammenhang zwischen der Anzahl der Regeln und der Paketdurchsatzrate, was einem reinen Leistungstest entspricht. Getestet wurden verschiedene Szenarien und die Leistung bei 10, 100 und 1000 MBit/s. Die wenigsten Testfälle berücksichtigten auch zustandsbehaftete Filterung. Zur Durchführung der Untersuchung wurde ein Framework mit erweiterbaren Testfällen entwickelt, das auf dem Testsystem ein Regelwerk einstellt, Pakete schickt

und sie bei Eintreffen auf Unterschiede vergleicht.

Firewallbetrieb – Penetrations- und Produktivumgebungstests

Haeni [Hae97] gibt in seiner Arbeit eine Anleitung und Empfehlungen für das Vorgehen bei Penetrationstests in Produktivumgebungen. Großes Augenmerk legt er dabei auf die Festlegung des durchführenden Testers, der daraus resultierenden Vertrauensstellung und Ergebnisqualität. Weder die Hersteller, noch rekrutierte Hacker stellen für ihn eine geeignete Auswahl dar und er empfiehlt ausgebildete und erfahrene Sicherheitsexperten damit zu beauftragen. Die Methodologie selbst gliedert er in vier Phasen: die indirekte und die direkte Erfassung von Informationen sowie die externen und die internen Angriffe auf das Zielobjekt. Der Bericht ist eher praktisch orientiert, beschreibt aber sehr gut die typische manuelle Durchführung einer Sicherheitsuntersuchung.

Vigna [Vig97] dagegen kritisiert die übliche Testpraxis bei Firewallprodukten und deren Einsatz. Die oft angewandten Checklisten-basierenden Produkttests, Penetrationstests oder gar Überprüfungen für die Vergabe von Sicherheitszertifikaten, wie sie die NCSA ausstellt, folgen alle der „test once – deploy many“ Strategie, die in einer Produktivumgebung ihre Aussagekraft verliert. Deswegen empfiehlt er Tests ausschließlich unter Berücksichtigung der konkreten Netzwerkstruktur einer Produktivumgebung (field testing) durchzuführen. Die eigentliche Überprüfung basiert auf einem formalen Modell, das die Verbindungen zwischen den Rechnern und der Firewall in Graphen abbildet und Funktionen vereinbart, die Teilmengen der Knoten mit gewünschten Eigenschaften (Netzwerkdienste bis Ebene 4 des OSI-Modells) liefern. Darauf aufbauend kann er die beste Position einer Firewall in der untersuchten Topologie bestimmen bzw. die untersuchte Position bewerten.

Firewallkonfiguration – Analyse und Erstellung von Regeln und Richtlinien

Basis für die meisten Arbeiten ist oft *Firmato*, ein Firewall Management Toolkit, das die folgenden Aufgaben vereinfachen sollte: die Sicherheitsrichtlinie unabhängig von einem konkreten Produkt, von dessen Eigenschaften und von der Netzwerktopologie zu formulieren; die Firewallkonfiguration automatisch aus dieser Richtlinie zu generieren und die Möglichkeit schaffen Konfigurationen auf einer abstrakteren Ebene zu untersuchen. Dazu wurden ein Entity-Relationship-Model (ERM), eine Model Definition Language (MDL), ein Model Compiler und eine Visualisierung für die Regeln entwickelt [siehe Bar99].

Aufbauend auf den Arbeiten an Firmato hat die Forschungsgruppe um Wool *FANG*, die Firewall ANalysis Engine [MWZ00], entwickelt. Auf diesem Prototyp aufbauend entstanden dann der *Lumeta Firewall Analyzer* (LFA) [Woo01; MWZ05] und schließlich eine kommerzielle Version von AlgoSec[3]. Alle Versionen sind in der Lage automatisch verschiedene Konfigurationssprachen und routing-Informationen

[3]Informationen zum AlgoSec Firewall Analyzer können unter `http://www.algosec.com` abgerufen werden.

einzulesen, um daraus einen Bericht über potentielle Sicherheitsprobleme zu erstellen. Ebenfalls von A.Wool stammt eine manuelle Analyse von typischen Konfigurationsfehlern in produktiven Umgebungen [Woo04], bei der er prinzipiell Handlungsbedarf und Verbesserungsmöglichkeiten durch Einsatz von geeigneten Konfigurations- und Testwerkzeugen sieht.

Der *Firewall Policy Advisor* (FPA) ist ein grafisches Werkzeug, das die konflikt- und fehlerfreie Verwaltung von Firewallregeln unterstützen soll [ASH03]. Darin enthalten sind Algorithmen zur automatischen Entdeckung von Anomalien und Konflikten in bestehenden Firewallregelwerken sowie Regelwerkmodifikationen. Unter Anomalien verstehen die Autoren Regeln, die sich überdecken, in Beziehung zu einander stehen, sich komplett widersprechen oder redundant sind. Aufbauend auf diesen Erkenntnissen erweiterten die Autoren die Funktionalität und die zugrunde liegenden Techniken auf verteilte Firewalls und beschreiben sie in [ASH04]. Dank dieser Arbeit ist es zwar möglich Richtlinien zu entwerfen und zu überprüfen, die Umsetzung in ein reales Regelwerk muss jedoch weiterhin manuell durchgeführt werden, was u.U. eine neue Fehlerquelle einführt und keine Aussage über die tatsächliche Wirkung des Regelwerks trifft.

FIREMAN steht für das FIREwall Modelling and ANalysis toolkit und beherrscht die statische Analyse von Firewallkonfigurationen [Yua06]. Diese sehr aktuelle Arbeit baut auf den vorherigen Arbeiten (u.a. A.Wool) auf und kombiniert sie. FIREMAN eignet sich zur Analyse von einzelnen wie auch verteilten Firewall-Lösungen. Es kann echte Konfigurationen parsen und in eine eigene interne BDD[4]-basierende Darstellung umwandeln. Diese kann es auf Inkonsistenzen und Ineffizienzen innerhalb einer Firewall, zwischen den Firewalls und bei mehreren möglichen Pfaden untersuchen. Zudem werden Abweichungen zwischen den Konfigurationen und den Sicherheitsrichtlinien anhand der vom Administrator erweiterbaren white- und blacklists verglichen.

FACE, die Firewall Analysis and Configuration Engine [VP05], ist ein Werkzeug, das hilft in einer verteilten Firewallumgebung die Sicherheitsrichtlinien korrekt zu verteilen. Das Hauptaugenmerk liegt auf der Implementierung von vertrauenswürdigen Knoten(-wolken), in denen Adressfälschung nicht möglich ist und daraus die Reduzierung von unnötigem Netzwerkverkehr resultiert. Nicht implementierbare Richtlinien sollen einfach identifiziert werden können und der „firewall analyzer“ kann Anfragen nach erlaubtem Verkehr, Korrektheit und Konsistenz des Regelwerks beantworten sowie die Folgen von Konfigurationsänderungen oder der Kompromitierung eines Systems/einer Firewall aufzeigen. Das Modell orientiert sich an netfilter/iptables und erlaubt als eines der wenigen Modelle auch Zustände und TCP-Flags zu berücksichtigen.

„Filtering postures“ [Gut97] stellt eine einfache Lisp- bzw. Scheme-ähnliche Sprache für das Beschreiben von globalen Netzwerkzugriffsrichtlinien vor. Dazu wird ein Algorithmus gezeigt, der für eine gegebene Netzwerktopologie die Regeln für die einzelnen Router errechnet. Dadurch soll gewährleistet werden, dass die Regeln die Richtlinien korrekt umsetzen. Zudem wird eine optimale und redundanzfreie Vertei-

[4]BDD steht für *Binary Decision Diagram*.

lung der Policy auf die Router und Firewalls in einer verteilten Umgebung erreicht. Leider wird auf die Umsetzung auf den konkreten Geräten nicht eingegangen.

In „An expert system for analyzing firewall rules“ [EZ01] wurde ein *Expertensystem* basierend auf Constraint Logic Programming (CLP) entwickelt, mit dem sich ein Regelwerk[5] analysieren lässt. Das Expertensystem enthält eine „Datenbank“ mit Fakten über verschiedene Netzwerkprotokolle, deren Sicherheitsmerkmale und häufige Fehlkonfigurationen. Es ist möglich Anfragen bezüglich einer eingelesenen Konfiguration zu stellen, die die erwartete Reglementierung, erlaubte bzw. verbotene Verbindungen oder die Richtlinie bzgl. eines Paketes enthält. Die in der Arbeit vorgestellten Fakten kennen zwar Protokollmerkmale bis hin zur Schicht 4, aber keine zustandsbehaftete Filterung oder Paketflussveränderungen wie NAT. Auch werden nur einfache Regeln mit accept/deny und einer geordneten Regelliste unterstützt.

Gouda und Liu [GL05b] beschäftigen sich mit dem Problem der Aufdeckung aller redundanten Regeln einer Konfiguration. Zunächst werden die Voraussetzungen für die Entdeckung aller solcher Regeln vorgestellt, aufgrund derer die Regeln in aufwärts-redundante und abwärts-redundante Regeln klassifiziert werden. Schließlich werden auf *firewall decision trees* (FDT) arbeitende Methoden vorgestellt, die beide Redundanztypen entdecken können. Die Arbeit zielt auf die Optimierung eines Regelwerks und bleibt abstrakt auf der Modellebene ohne eine konkrete Anwendung vorzustellen.

ITVal ist ein Open Source Werkzeug, das die Konfigurationsanalyse von netfilter/iptables Firewalls, ähnlich der Funktionalität von FANG und LFA, ermöglicht. Die Basis von ITVal ist eine Funktionsbibliothek für die effiziente Manipulation von multi-way decision diagrams, die zur Darstellung der Regeln und der Anfragen über die Konfiguration genutzt werden. Neben der Betrachtung des Designs und der Implementierung von ITVal werden auch Beispiele vorgestellt, wie häufige Konfigurationsfehler entdeckt und korrigiert werden können [vgl. MK05].

Firewallmodellierung

Von Gouda und Liu stammt auch das erste Modell für zustandsbehaftete Firewalls [GL05a]. Die Basis des Modells ist die Auftrennung einer zustandsbehafteten Firewall in zwei Bereiche – einen zustandsbehafteten und einen zustandslosen Bereich. Im ersten Bereich werden Informationen über bereits akzeptierte Pakete gespeichert, die als Zustand der Firewall bezeichnet werden. Beim Eintreffen eines Paketes wird es um eine zusätzliche Markierung, die aus dem aktuellen Zustand der Firewall berechnet wird, erweitert. Im nächsten Schritt wird das Paket incl. der Markierung mit den Regeln des zustandslosen Bereiches verglichen, um die erste zutreffende Regel zu finden, die eine Reglementierung vorgibt. Dieses Modell hat sich trotz seiner Einfachheit als besonders geeignet erwiesen und kann auch eine Vielzahl von Zustandsüberwachungsverfahren abbilden. Es baut auf vielen Erfahrungen mit zustandslosen Firewalls sowie deren Analysen auf und ist zu ihnen kompatibel, so dass

[5] In der angegebenen Quelle werden prototypisch Cisco ACLs unterstützt.

sie auch mit diesem Modell spezifiziert werden können. Der Bericht stellt neben dem Modell selbst auch eine Analyse der Eigenschaften von zustandsbehafteten Firewalls vor. Es werden u.a. eine Methode sowie Kriterien vorgestellt, mit denen sich überprüfen lässt, ob eine Firewall in der Tat zustandsbehaftet (truly stateful) ist.

Spezifikationsbasierendes Testen

Jürjens und Wimmel [JW01] schlagen eine Methode für das spezifikationsbasierte Testen von Firewalls vor. Sie erlaubt es die Firewalls zusammen mit dem umgebenden Netzwerk formal zu definieren und daraus automatisch Testfälle abzuleiten, die die Firewall auf Schwachstellen (vulnerabilities) überprüfen können. Durch das eingesetzte CASE-Werkzeug ist die Methode flexibel und einfach zu benutzen. Fragwürdig an dieser Arbeit ist jedoch wann und von wem diese CASE-Spezifikation erzeugt und vor allem synchron mit der Netzwerkumgebung gehalten werden soll. Auch wird nicht beschrieben, wie die Pakete zur Überprüfung der Testfälle erzeugt, übertragen und getestet werden. Aus Sicht des Autors fehlt auch eine Stellungnahme zum Testen von zustandsbehafteten Paketfiltern.

Untersuchung der Sicherheitsrichtlinien

Diana von Bidder-Senn [BS06] zeigt eine formelle Beschreibung für die Sicherheitsrichtlinien eines Netzwerks auf, aus der automatisch Testfälle generiert werden können, die ein eingesetztes System testen können. Somit wird die Konformität eines Regelwerks zu einer aufgeschriebenen Richtlinie überprüft. Die Sicherheitsrichtlinie wird im Sinne eines Zugriffsmodells, also *wer* darf *wo* auf *was* zugreifen, verstanden. Die generierten Testfälle hängen stark von der organisationsspezifischen Umgebung ab, können aber laut Von Bidder-Senn Fehler sowohl in der Konfiguration selbst wie auch in der eingesetzten Firewall aufspüren. Bei den Tests wird die Firewall als Black-Box betrachtet und die Protokollflüsse allein aufgrund der Spezifikation der Kommunikation zwischen den Endpunkten hergestellt. Für die Betrachtung der Eignung dieses Ansatzes wurde aus der Protokollspezifikation für TCP ein Automat für den Zwischenpunkt abgeleitet, mit dem drei ausgewählte Paketfilter auf Übereinstimmung verglichen wurden. Dabei wurden jeweils Abweichungen festgestellt, die auf die Interpretation der Schutzfunktion des Paketfilters durch den jeweiligen Anbieter zurückgeführt werden konnten. Der entwickelte TCP-Automat kann durch das Hinzufügen oder Entfernen von Transitionen auf ein konkretes Produkt eingestellt werden.

In [IHAS05] wird eine neuartige Technik für das effiziente und automatische Testen von Firewalls vorgestellt, die spezielle Testfälle unter Berücksichtigung der eingesetzten Sicherheitsrichtlinie erstellen und testen kann. Die Richtlinien-basierende Segmentierung des Adressraumes kann die Testfallgenerierung auf potentiell fehlerhafte Regionen in dem von der Firewall genutzten Eingaberaum reduzieren bzw. konzentrieren, was das Problem des automatischen Testens handhabbar macht und einen signifikant höheren Vertrauensgrad im Vergleich zum zufälligen Testen gibt.

10.3. Vergleich

Aus der Sicht eines vollständigen Tests entsprechen Penetrationstests dem „blinden Herumstochern“ in einer unbekannten Konfiguration mit der Absicht ausnutzbare Schwachstellen zu finden [Bsp. Hae97]. Auch systematische Tests in einer einzigen statischen Umgebung, wie sie für verschiedene Zertifizierungen durchgeführt werden, verlieren in einem realen Netzwerk mit einer konkreten Konfiguration ihre Aussagekraft [vgl. Vig97].

Die Überprüfung der Qualitätsmerkmale Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartungsfreundlichkeit und Übertragbarkeit bei der Firewallsoftware ist für Entwickler und Hersteller von großer Bedeutung. Für die Linux Firewall netfilter/iptables wurde je ein Testansatz für Regressionstests [DH04] und für Leistungstests [HPS03] vorgestellt.

Ein Endbenutzer oder ein Administrator ist jedoch daran interessiert den Betrieb und die Umsetzung der Aufgabenstellung zu überprüfen. Hierzu wurden verschiedene Arbeiten zur Analyse, Modifikation und zum Testen von (formalen) Sicherheitsrichtlinien und Regelwerken vorgestellt. Dabei konnten vier Ansätze identifiziert werden:

- Analyse der Richtlinie oder eines abstrakten Regelwerks,
- Analyse der Richtlinie – abgeleitet aus den Regeln,
- Erstellen der Regeln – abgeleitet aus der Richtlinie,
- Testen nach Ableitung der Testfälle aus der Richtlinie oder den Regeln.

Der erste Fall birgt das Problem, dass Richtlinien und das Regelwerk separat synchron gehalten oder sogar manuell ineinander überführt werden müssen. Zudem setzt es voraus, dass eine informale Sicherheitsrichtlinie für eine bestimmte Umgebung, etwa eine Firma oder ein Netzwerk, bereits existiert und in eine formale Form überführt werden kann. Außerdem umfasst eine Sicherheitsrichtlinie die globale Sicht – also mehrere Objekte, die alle einzeln identifiziert, spezifiziert und getestet werden müssen.

Die Ableitung der Regeln aus einer Richtlinie setzt letztere ebenfalls voraus. Da eine Richtlinie allgemein und abstrakt definiert ist, müssen firewallspezifische Eigenschaften und Funktionen trotzdem manuell in die erstellte Konfiguration eingefügt werden, was die Benutzbarkeit und die Übereinstimmung der beiden einschränkt.

Werden Firewalls aber schon eingesetzt, so werden ihre Konfigurationen oftmals die ersten tatsächlich vorhandenen formalen Beschreibungen der netzwerkbezogenen Sicherheitsrichtlinien sein. Diese lassen sich sowohl zu einer Richtlinie verallgemeinern, formal analysieren wie auch in eine Teststellung abbilden.

Gleich mehrere Werkzeuge und Arbeiten behandeln die Überführung der konkreten Regeln in eine abstrakte Form, eine interne Darstellung oder eine allgemeine Richtlinie. FANG, der Lumeta Firewall Analyzer und der AlgoSec Firewall Analyzer analysieren das Regelwerk auf potentielle Sicherheitsprobleme, dh. evtl. unerwünschte Kommunikationskanäle [MWZ00; Woo01; MWZ05]. Eine Analyse auf Anomalien

oder Inkonsistenzen des Regelwerks wird nicht durchgeführt. Einen ähnlichen Ansatz verfolgen die Arbeit von Eronen und Zitting [EZ01] und das Tool ITVal [MK05]. FIREMAN [Yua06] kann neben einzelnen auch verteilte Firewallkonfigurationen auf Anomalien überprüfen.

Ebenfalls mehrere Arbeiten untersuchen die formale Richtlinie selbst oder eine Form eines abstrakten Regelwerks. Mit FPA [ASH03] oder mit den in [GL05b] vorgestellten Algorithmen können Anomalien und Redundanzen in einer vereinfachten Notation der Sicherheitsrichtlinien bzw. des Regelwerks gefunden werden und gegebenenfalls Korrekturvorschläge gemacht werden. FPA kann zusätzlich Modifikationen auf der internen BDD-basierenden Darstellung interaktiv durchführen. „Filtering postures“ [Gut97] und FACE [VP05] konzentrieren sich dagegen auf die richtlinienkonforme Verteilung der Konfiguration auf die Knoten einer verteilten Firewall und die Analyse der möglichen Zugriffspfade.

Firmato wurde als einziges vorgestelltes Werkzeug für die Verwaltung und die Überführung einer Richtlinie in eine konkrete Konfiguration entwickelt [Bar99].

Am ähnlichsten zu der vorliegenden Arbeit ist die Untersuchung von Specification-based Firewall Testing [BS06], die sich mit der Abbildung einer Sicherheitsrichtlinie in Testfälle unter Berücksichtigung der Zielfirewall und des Netzwerks beschäftigt. Die dafür notwendige Überführung der Richtlinien in ein Regelwerk wird erwähnt, aber nicht gelöst. Durch die Ableitung der Testserien aus der allgemeinen Richtlinie werden keine Eigenheiten der jeweiligen IUT berücksichtigt und sind auf einen sehr einfachen Paketfilter beschränkt. Paketveränderungen oder NAT können nicht überprüft werden. Stattdessen muss den Testgeneratoren eine Beschreibung der spezifischen TCP-Zustandsmaschine zugeführt werden, mit welcher dann für jeden Testpunkt als Repräsentation aller spezifizierten Protokollflüsse die Zustandsübergänge vollständig getestet werden. Dies führt zu einer sehr großen Test- und Paketanzahl sowie, auch bedingt durch eine fehlende Berücksichtigung der Zustandsverfolgung, zu manchen *false positives* und *false negatives*.

Die Betrachtung der Protokollflüsse von den Endpunkten aus und die darauf basierende Überprüfung der Konformität kann die Frage beantworten, ob ein bestimmter Protokollfluss von einem Paketfilter zugelassen oder unterdrückt wird.

Um die Frage zu beantworten, welchen Einfluss eine Konfiguration des Paketfilters auf die zugelassenen oder unterdrückten Protokollflüsse hat, müssten sämtliche denkbaren Protokollfüsse, die im Paketfilter potenziell konfigurierbar sind, getestet werden. Die Überprüfung des TCP-Automaten im Paketfilter wird zur Bewertung der allgemeinen Konformität zur Protokollspezifikation durchgeführt. Eine Aussage über die Konfiguration wird nicht getroffen.

Die vorliegende Arbeit geht einen neuen Weg, bei dem die produktspezifischen Regeln zunächst vollständig in eine abstrakte Form überführt und dann analysiert werden. Die Testbeschreibung basiert auf den Analyseergebnissen. Die Fragestellung der Tests bezieht sich auf eine Aussage über den Einfluss einer Paketfilterkonfiguration auf die zugelassenen oder unterdrückten Protokollflüsse. Deshalb wird ausgehend von der Konfiguration bzw. deren Repräsentation die Sichtweise des Paketfilters auf den Protokollfluss eingenommen, um gezielt die in der Konfiguration beschriebenen

und vom Paketfilter untersuchten Merkmale des Protokollflusses herzustellen und mit der postulierten Reaktion zu vergleichen. Genau wie Von Bidder-Senn (Kap. 5.1 und 5.3) wurde dabei festgestellt, dass sich die Sicht des Paketfilters auf die Protokollflüsse von der strikten Auslegung der Protokollspezifikation unterscheidet und diese Abweichungen nicht (formal) dokumentiert sind. Deswegen müssen sie rekonstruiert werden. Die Analyse der ausgewählten Paketfilter und die darauf aufbauende Entwicklung eines Modells zum Beschreiben des Paketfilter in der vorliegenden Arbeit stellen die Voraussetzung für einen Test zum Überprüfen der Paketfilterkonfiguration dar. Das entwickelte Werkzeug FWTStrategy setzt eine solche Teststrategie prototypisch um.

Als vergleichbare Testwerkzeuge kämen Blowtorch, FTester und fwtest von der ETH Zürich in Frage. Blowtorch bietet ein mächtiges, aber auch komplexes Framework an, was sich durch die Implementierung in der Sprache C++ nur schwer vom Endbenutzer anpassen ließe. Leider konnte auch neben dem vorgestellten Bericht keine weitere Quelle für eine Evaluation gefunden werden. FTester ist zwar in einer Scripting-Sprache implementiert, ist aber nur durch eine statische Konfigurationsdatei zu steuern und erlaubt nur sehr einfache Manipulationen des Paketes. Ähnlich verhält es sich mit fwtest von der ETH Zürich. Es sind nicht alle Protokollfelder manipulierbar und keine Nutzdaten definierbar, es ist keine bedingte Ablaufsteuerung möglich, es werden nur die versendeten oder keine Pakete erwartet, und die Abweichungen zwischen den gesendeten und den empfangenen Paketen können nur grob definiert werden.

Die in dieser Arbeit entwickelte FWTStrategy ist als Werkzeug flexibler einzusetzen und mächtiger in der zur Verfügung gestellten Funktionalität. Der Testablauf ist nicht fest durch eine Konfiguration oder einen Ablaufplan vorgegeben, sondern wird durch die Firewallbeschreibung und die Konfiguration bestimmt. Die Tests können bereits im Prototyp mehrere Aktionen der Firewall auswerten und sind nicht nur auf ACCEPT oder DROP beschränkt. Die Pakete werden anhand der vorgegebenen Testvektoren erstellt und bei Empfang des Paketes mit dem gesendeten Paket verglichen. Paketveränderungen können erkannt und bewertet werden. Damit wurde auch die Unterstützung für das Testen von NAT umgesetzt. Ebenfalls ausgewertet werden eventuelle Fehlermeldungen der vermittelnden Knoten, die auf eine Korrelation zu den gesendeten Paketen überprüft werden. Damit ist es möglich verschiedene REJECT-Arten und Fehlernachrichten zu erkennen.

Die erreichte Testabdeckung ist zwar bezogen auf den gesamten Testraum ebenfalls unvollständig, sie geht aber aufgrund der Orientierung an der Modellierung des Paketfilters und der Konfiguration weitaus strukturierter und gezielter vor als die anderen Ansätze. Für die in der Analyse identifizierten Testvektoren kann jedoch bei einer entsprechenden Erweiterung der Teststrategie eine vollständige Testabdeckung erreicht werden. Dabei wird durch die Teststrategie gleichzeitig versucht möglichst effizient vorzugehen und redundante Tests zu reduzieren. Der Einsatz der FWTStrategy und die Interpretation der Testergebnisse sind für einen fachkundigen Benutzer ausgelegt, der kein Experte sein muss. Dadurch wird eine erhöhte Benutzerfreundlichkeit des Testwerkzeugs erreicht und ein breiterer Benutzerkreis angesprochen.

11. Zusammenfassung und Ausblick

Die vorliegende Arbeit untersucht die Fragestellung, wie sich die Konfiguration eines Paketfilters auf die von ihm akzeptierten bzw. abgelehnten Protokollflüsse auswirkt und ob der Paketfilter sich konform zu den Protokollspezifikationen verhält.

Der verfolgte Ansatz zur Beantwortung der Frage ist die Entwicklung einer allgemeinen Teststrategie für zustandsbehaftete Paketfilter. Bei der Betrachtung der Untersuchungsobjekte wurde in Kapitel 2 festgestellt, dass es verschiedene Teilfunktionen gibt und die jeweiligen Produkte sich in der Komposition dieser Funktionen unterscheiden. Für die Eingrenzung der Untersuchung wurden die verschiedenen Firewalltypen kategorisiert und die weitere Betrachtung auf sechs für die Marktverteilung repräsentative Paketfilter eingeschränkt.

In Kapitel 3 wurden die Funktionen der Firewalls in Gruppen eingeordnet, die aus der Analyse von Sicherheitszieles abgeleitet wurden. Auch bei ähnlichen angebotenen Funktionen wurden in der internen Umsetzung Unterschiede festgestellt. Dies konnte darauf zurück geführt werden, dass es keine Musterfirewall gibt, an der sich alle orientieren können, sondern vielmehr unterschiedliche Architekturen und Strategien eingesetzt werden. Gemeinsam ist allen jedoch die ähnliche äußere Schnittstelle, also das Verhalten bei der Vermittlung und der Reglementierung der Protokollflüsse. Ausgehend von der Vielfältigkeit der Firewalls mussten die internen Abläufe und Strukturen analysiert werden.

Bei der Betrachtung der Konfigurierbarkeit der Paketfilter wurde die Struktur der Regelwerke, die Möglichkeiten zur Strukturierung des Regelwerks und zur Erhöhung der Wartbarkeit sowie die Mächtigkeit der Paketfilter und der Paketveränderungen untersucht. Dabei wurde der Paketfilter netfilter/iptables als die flexibelste Lösung identifiziert, mit dem sich ein Großteil der Funktionen der anderen Paketfilter vergleichbar realisieren lassen. Aus diesem Grund wurde dieser für die weitere Modellierung und Umsetzung der Teststrategie ausgewählt. Die Filterkriterien und die Filteraktionen von iptables wurden zu funktionell vergleichbaren Gruppen zusammengefasst und die Relevanz der Merkmale für eine allgemeine Repräsentativität bewertet (vgl. Abschnitt 3.3).

In Kapitel 4 wurde die Funktionsweise der Paketfilter weiter analysiert. Dazu wurden speziell die Paketflüsse und die zustandsbehaftete Verbindungsverfolgung der Paketfilter anhand der Dokumentation und Quellcodes rekonstruiert, um die Abhängigkeiten der Funktionen und ihre Wirkung auf den Entscheidungsprozess beim Filtern festzustellen. Dieser Schritt war notwendig, weil die internen Mechanismen der Firewalls nicht oder nur ungenügend dokumentiert sind und sie aufgrund ihrer beobachtenden Position zwischen den Endpunkten Entscheidungen treffen müssen, die von der Sicherheitsrichtlinie vorgegeben werden. Ein Paketfilter

hat eine eigene sicherheitsbezogene Sicht auf den Protokollfluss, weswegen nicht die Endpunkt-basierenden Protokollspezifikationen für diese Analyse herangezogen werden konnten. Zudem ist die Erfassung der Kernfunktion der zustandsbehafteten Paketfilter und die Identifizierung des Konfigurationseinflusses auf das Verhalten Voraussetzung für einen gezielten Test der Paketfilter. Hier konnte festgestellt werden, dass die untersuchten Paketfilter vergleichbare Informationen zur Identifizierung der Verbindungen speichern.

Kapitel 5 fasst die Analyseergebnisse schließlich zusammen und entwickelt ein Modell zur Beschreibung der Paketflüsse und der Entscheidungspunkte für die Paketfilterung und die Paketveränderung. Das Modell ist nach dem Baukastenprinzip aufgebaut, mit dem durch die geeignete Verknüpfung der identifizierten Bausteine der Paketfluss und die Abhängigkeiten der Funktionen für einen allgemeinen Paketfilter beschrieben werden können. Die Bausteine können den Gruppen Funktionen, Arbeitsobjekte, Zustandsdaten und Zustandsfunktionen zugeordnet werden (vgl. Abschnitt 5.1). Für die Modellierung der Funktionen wurden zwei weitere Kategorisierungen eingeführt, die den Zweck bzw. die Entscheidungsart der Funktionen erfassen.

Durch die Analyseergebnisse der Firewalls und Paketfilter stehen nun mehrere Klassifizierungen zur Verfügung, mit denen sich Firewalltypen und Firewallfunktionen beschreiben und vergleichen lassen. Die neue Modellierung wurde beispielhaft auf die Funktionen und Paketflüsse des netfilter/iptables-Paketfilters angewendet (vgl. Abschnitt 5.2 und Abschnitt 5.3).

Kapitel 6 legt die methodischen und theoretischen Grundlagen für die Entwicklung einer Teststrategie für die Überprüfung der gesetzten Ziele. Als Basis wurden zwei anerkannte Standards zum Testen der Konformität von Protokollen und Netzwerksystemen, ISO9646 bzw. X290, eingeführt. Weiterhin wurde die Testbarkeit von Protokollen und Protokollflüssen untersucht. Dabei wurde festgestellt, dass die Komplexität, die die heutigen Firewallsysteme bieten, mit den zur Verfügung stehenden Ressourcen nicht mehr vollständig getestet werden kann. Dafür wurden Strategien zur Reduzierung der Komplexität und Testraumgröße erarbeitet.

Die Teststrategie wird in Kapitel 7 entwickelt. Dabei wurden die Ziele der Unabhängigkeit von konkreten Produkten, Unabhängigkeit von den betrachteten Protokollen, Einstellbarkeit der Tests für eine konkrete Firewall, Bestimmung der Relevanz und der Detailtiefe der Testfälle, Optimierung der Nutzung der Testressourcen, Durchführbarkeit der Tests unter Labor- und Realbedingungen sowie die Wiederholbarkeit der Tests verfolgt. Die Ergebnisse in Kapitel 9 zeigen, dass die Teststrategie vielfältig einsetzbar ist und bereits im Prototyp grundlegende Konfigurationen überprüfen kann. Durch den modularen Aufbau und die Wahl einer einfach zu erlernenden Hochsprache für die Umsetzung ist die einfache Erweiterbarkeit der Teststrategie für weitere Testumgebungen und Anforderungen gegeben.

Die Bewertung und der Vergleich der Teststrategie im Kontext anderer Werkzeuge und Arbeiten in Abschnitt 10.3 zeigt den Beitrag dieser Arbeit zur Verbesserung und zur Erleichterung der Testbarkeit von Firewallkonfigurationen. Dies wird in einem benutzerfreundlichen Werkzeug mit Nutzen für Entwickler und Administratoren zusammengefasst.

Ausblick

Der entwickelte Prototyp der Teststrategie bietet eine gute Grundlage für weitere Untersuchungen und das Testen von Firewalls. Die modulare Struktur wurde von Anfang an dafür konzipiert Erweiterungen zu entwickeln und einzubinden. Aufgrund der beschränkt verfügbaren Bearbeitungszeit konnten nur ausgewählte Funktionen umgesetzt werden, die aber die Grundlage jeder Konfiguration darstellen und die Möglichkeiten der Teststrategie aufzeigen.

Für die Fortführung der vorliegenden Arbeit bietet es sich zunächst an die fehlenden Filterkriterien und Filteraktionen des netfilter/iptables Paketfilters zu vervollständigen. Damit könnte die Testabdeckung für diesen Paketfilter erhöht werden und die Mächtigkeit der Programmschnittstellen überprüft werden.

Die Untersuchung beschränkte sich auf die Umsetzung einer Teststrategie für die Protokolle TCP, UDP und ICMP – ohne die Anwendungsprotokolle zu berücksichtigen. Damit können keine Protokollflüsse für den related-Zustand bei TCP und UDP hergestellt werden. Die Erweiterung der Teststrategie auf die Herstellung von Protokollflüssen der Anwendungsebene bedarf der Entwicklung einer neuen Schnittstelle, die auf Basis der implementierten Steuerung der Netzwerk- und Transportschicht auch die Nutzlast der Anwendungsschicht steuern, testen und auswerten kann. Eine mögliche Fortführung der Arbeit könnte die Eignung von Agenten oder Automaten für die Realisierung der Protokollinstanzen untersuchen.

Die Teststrategie baut zur Stimulation des Probanden auf dem Programm FWTest auf. Damit ist eine Kommunikation von zwei Protokollpartnern durch die Firewall möglich. Die aktuelle Architektur der FWTest-Agenten schränkt deren Betrieb auf Linux-Systeme ein, wodurch nur der vermittelnde Aspket untersucht werden kann. Dadurch entzieht sich der zur eingehende und von der Firewall ausgehende Verkehr der Untersuchung. Der Betrieb der Agenten auf einer Firewall oder Firewall-Appliance bedarf einer Modifizierung der Agenten und sollte weiter verfolgt werden.

Schließlich wird die Erweiterung der Fähigkeiten des FWAs die Ausweitung der Tests auf andere Paketfilter ermöglichen. Dafür muss eine Beschreibung und Modellierung der anderen Paketfilter vorliegen, die in einem Projekt bearbeitet werden könnte.

Anhang A.

Anhang

A.1. Modellierung von netfilter/iptables

phase	chain	task	type	next processing point
PREROUTING	raw	ACCEPT	forward	PREROUTING:add_state
PREROUTING	raw	DROP	stop	-
PREROUTING	raw	LOG	non-term	continue
PREROUTING	raw	NOTRACK	forward	PREROUTING:mangle
PREROUTING	raw	ULOG	non-term	continue
PREROUTING	add_state	add_state	forward	PREROUTING:mangle
PREROUTING	mangle	ACCEPT	forward	PREROUTING:nat
PREROUTING	mangle	CONNMARK	non-term	continue
PREROUTING	mangle	CONNSECMARK	non-term	continue
PREROUTING	mangle	DROP	stop	-
PREROUTING	mangle	DSCP	non-term	continue
PREROUTING	mangle	ECN	non-term	continue
PREROUTING	mangle	LOG	non-term	continue
PREROUTING	mangle	MARK	non-term	continue
PREROUTING	mangle	SECMARK	non-term	continue
PREROUTING	mangle	TOS	non-term	continue
PREROUTING	mangle	TTL	non-term	continue
PREROUTING	mangle	ULOG	non-term	continue
PREROUTING	nat	ACCEPT	forward	KERNEL:route
PREROUTING	nat	DNAT	forward	KERNEL:route
PREROUTING	nat	DROP	stop	-
PREROUTING	nat	LOG	non-term	continue
PREROUTING	nat	NETMAP	forward	KERNEL:route
PREROUTING	nat	REDIRECT	forward	KERNEL:route
PREROUTING	nat	ULOG	non-term	continue
INPUT	filter	ACCEPT	forward	INPUT:mangle
INPUT	filter	DROP	stop	-
INPUT	filter	LOG	non-term	continue
INPUT	filter	REJECT	stop	-
INPUT	filter	ULOG	non-term	continue
INPUT	mangle	ACCEPT	forward	KERNEL
INPUT	mangle	CONNMARK	non-term	continue
INPUT	mangle	CONNSECMARK	non-term	continue
INPUT	mangle	DROP	stop	-
INPUT	mangle	DSCP	non-term	continue
INPUT	mangle	ECN	non-term	continue
INPUT	mangle	LOG	non-term	continue
INPUT	mangle	MARK	non-term	continue
INPUT	mangle	SECMARK	non-term	continue
INPUT	mangle	TOS	non-term	continue
INPUT	mangle	TTL	non-term	continue

phase	chain	task	type	next processing point
INPUT	mangle	ULOG	non-term	continue
FORWARD	mangle	ACCEPT	forward	FORWARD:filtering
FORWARD	mangle	CLASSIFY	non-term	continue
FORWARD	mangle	CONNMARK	non-term	continue
FORWARD	mangle	CONNSECMARK	non-term	continue
FORWARD	mangle	DROP	stop	-
FORWARD	mangle	DSCP	non-term	continue
FORWARD	mangle	ECN	non-term	continue
FORWARD	mangle	LOG	non-term	continue
FORWARD	mangle	MARK	non-term	continue
FORWARD	mangle	SECMARK	non-term	continue
FORWARD	mangle	TCPMSS	non-term	continue
FORWARD	mangle	TOS	non-term	continue
FORWARD	mangle	TTL	non-term	continue
FORWARD	mangle	ULOG	non-term	continue
FORWARD	filter	ACCEPT	forward	POSTROUTING:mangle
FORWARD	filter	DROP	stop	-
FORWARD	filter	REJECT	stop	-
FORWARD	filter	LOG	non-term	continue
FORWARD	filter	TCPMSS	non-term	continue
FORWARD	filter	ULOG	non-term	continue
OUTPUT	raw	ACCEPT	forward	OUTPUT:add_state
OUTPUT	raw	LOG	non-term	continue
OUTPUT	raw	TCPMSS	non-term	continue
OUTPUT	raw	NOTRACK	forward	OUTPUT:mangle
OUTPUT	raw	ULOG	non-term	continue
OUTPUT	add_state	add_state	forward	OUTPUT:mangle
OUTPUT	mangle	ACCEPT	forward	OUTPUT:nat
OUTPUT	mangle	CLASSIFY	non-term	continue
OUTPUT	mangle	CONNMARK	non-term	continue
OUTPUT	mangle	CONNSECMARK	non-term	continue
OUTPUT	mangle	DROP	stop	-
OUTPUT	mangle	DSCP	non-term	continue
OUTPUT	mangle	ECN	non-term	continue
OUTPUT	mangle	LOG	non-term	continue
OUTPUT	mangle	MARK	non-term	continue
OUTPUT	mangle	SECMARK	non-term	continue
OUTPUT	mangle	TCPMSS	non-term	continue
OUTPUT	mangle	TOS	non-term	continue
OUTPUT	mangle	TTL	non-term	continue
OUTPUT	mangle	ULOG	non-term	continue
OUTPUT	nat	ACCEPT	forward	OUTPUT:filter
OUTPUT	nat	DNAT	forward	OUTPUT:filter

phase	chain	task	type	next processing point
OUTPUT	nat	DROP	stop	-
OUTPUT	nat	LOG	non-term	continue
OUTPUT	nat	NETMAP	forward	OUTPUT:filter
OUTPUT	nat	REDIRECT	forward	OUTPUT:filter
OUTPUT	nat	TCPMSS	non-term	continue
OUTPUT	nat	ULOG	non-term	continue
OUTPUT	filter	ACCEPT	forward	KERNEL:route
OUTPUT	filter	DROP	stop	-
OUTPUT	filter	LOG	non-term	continue
OUTPUT	filter	REJECT	stop	-
OUTPUT	filter	TCPMSS	non-term	continue
OUTPUT	filter	ULOG	non-term	continue
POSTROUTING	mangle	ACCEPT	forward	POSTROUTING:nat
POSTROUTING	mangle	CLASSIFY	non-term	continue
POSTROUTING	mangle	CONNMARK	non-term	continue
POSTROUTING	mangle	CONNSECMARK	non-term	continue
POSTROUTING	mangle	DROP	stop	-
POSTROUTING	mangle	DSCP	non-term	continue
POSTROUTING	mangle	ECN	non-term	continue
POSTROUTING	mangle	LOG	non-term	continue
POSTROUTING	mangle	MARK	non-term	continue
POSTROUTING	mangle	SECMARK	non-term	continue
POSTROUTING	mangle	TCPMSS	non-term	continue
POSTROUTING	mangle	TOS	non-term	continue
POSTROUTING	mangle	TTL	non-term	continue
POSTROUTING	mangle	ULOG	non-term	continue
POSTROUTING	nat	ACCEPT	forward	POSTROUTING:finished
POSTROUTING	nat	DROP	stop	-
POSTROUTING	nat	LOG	non-term	continue
POSTROUTING	nat	MASQUERADE	forward	KERNEL
POSTROUTING	nat	TCPMSS	non-term	continue
POSTROUTING	nat	NETMAP	forward	KERNEL
POSTROUTING	nat	SNAT	forward	KERNEL
POSTROUTING	nat	ULOG	non-term	continue

Tabelle A.1: Tabellarische Modellierung der netfilter/iptables Firewall

Alle Schalter sind mit sysctl −w variable=value zu setzen bzw. mit sysctl variable abzufragen. Aus Platzgründen sind die hier gelisteten Schalter gekürzt wiedergegben und vor Gebrauch mit dem Prefix net.ipv4. zu ergänzen. Die Standardwerte sind dem Linux Kernel 2.6.20 entnommen.

Schalter	Standardwert	Beschreibung
icmp_echo_ignore_all	0	alle pings an FW stoppen
icmp_echo_ignore_broadcasts	1	broadcast pings an FW stoppen
icmp_errors_use_inbound_ifaddr	0	bei mehreren ifaces
icmp_ignore_bogus_error_messages	1	protokollverhalten
icmp_ratelimit	250	icmp msgs pro host/sec
icmp_ratemask	6168	ratelimit für ... (def: unreach, src-quench, time_excd, param_prob)
ip_default_ttl	64	
ip_local_port_range	32768,61000	min max für OUTPUT
ip_conntrack_max	65535	RAM-abhängig (hier: $\geq$ 128 MB)
ipfrag_low_thresh	196608	min Speicher für Fragmente
ipfrag_high_thresh	262144	max Speicher für Fragmente
ipfrag_time	30	max Wartezeit für Fragmente
ipfrag_max_dist	64	max Unordnung für eingehende Fragmente
tcp_max_syn_backlog	1024	max ausstehende SYN-ACKs
tcp_syncookies	0	
tcp_timestamps	1	
conf.all.accept_redirects	0	
conf.all.accept_source_route	0	
conf.all.forwarding	1	
conf.all.mc_forwarding	0	Weiterleitung von Multicast
conf.all.rp_filter	0	reverse-path filter
conf.all.secure_redirects	1	
conf.all.send_redirects	1	

Tabelle A.2: Auflistung von netfilter/iptables Optionen I.

Schalter	Standardwert	Beschreibung
netfilter.ip_conntrack_checksum	1	
netfilter.ip_conntrack_tcp_be_liberal	0	windows-tracking
netfilter.ip_conntrack_tcp_loose	3	start window-tracking nach n Pkts
netfilter.ip_conntrack_tcp_max_retrans	3	
netfilter.ip_conntrack_tcp_timeout_max_retrans	300	
netfilter.ip_conntrack_generic_timeout	600	
netfilter.ip_conntrack_icmp_timeout	30	
netfilter.ip_conntrack_tcp_timeout_close	10	
netfilter.ip_conntrack_tcp_timeout_close_wait	60	
netfilter.ip_conntrack_tcp_timeout_established	432000	
netfilter.ip_conntrack_tcp_timeout_fin_wait	120	
netfilter.ip_conntrack_tcp_timeout_last_ack	30	
netfilter.ip_conntrack_tcp_timeout_syn_recv	60	
netfilter.ip_conntrack_tcp_timeout_syn_sent	120	
netfilter.ip_conntrack_tcp_timeout_time_wait	120	
netfilter.ip_conntrack_udp_timeout	30	
netfilter.ip_conntrack_udp_timeout_stream	180	

Tabelle A.3: Auflistung von netfilter/iptables Optionen II.

A.2. Schnittstelle zum FWA

Ausgehend von den Eingaben vom FWA ist das Format der die Daten beinhaltenden Dateien festzuhalten. Aktuell ist dafür die Dateierweiterung .vec vereinbart worden. Diese Datei besteht aus fünf logischen Elementen: einer gemeinsamen Deklaration der Struktur der Vektoren in Form von Python Klassen, für jede abgebildete Tabelle der Firewall eine Liste von Vektoren (*_vl), eine Liste von möglichen Relationen zwischen den Vektoren (*_al) sowie eine Beschreibung des jeweiligen Netzwerks in Form von einer Liste mit den zulässigen Adressbereichen auf der A- und B-Seite der Firewall (*_tn). Die vorhandenen Tabellen und Beschreibungen sind in der Struktur Entries zusammengefasst.

```
class Entries(fwa.Entries):
  entries  = [ 'path', ]
  fw_type  = { 'name': 'iptables',      'version': '???' }
  fw_tool  = { 'name': 'iptables-save', 'version': '???' }
  analyzer = { 'name': 'fwa',           'version': '???' }
  vector   = { 'name': 'v3p',           'version': '???' }
  path     = { 'vl': path_vl, 'al': path_al, 'tn': path_tn}
```

Listing A.1: Hauptzugriffsstruktur einer Vektordatei

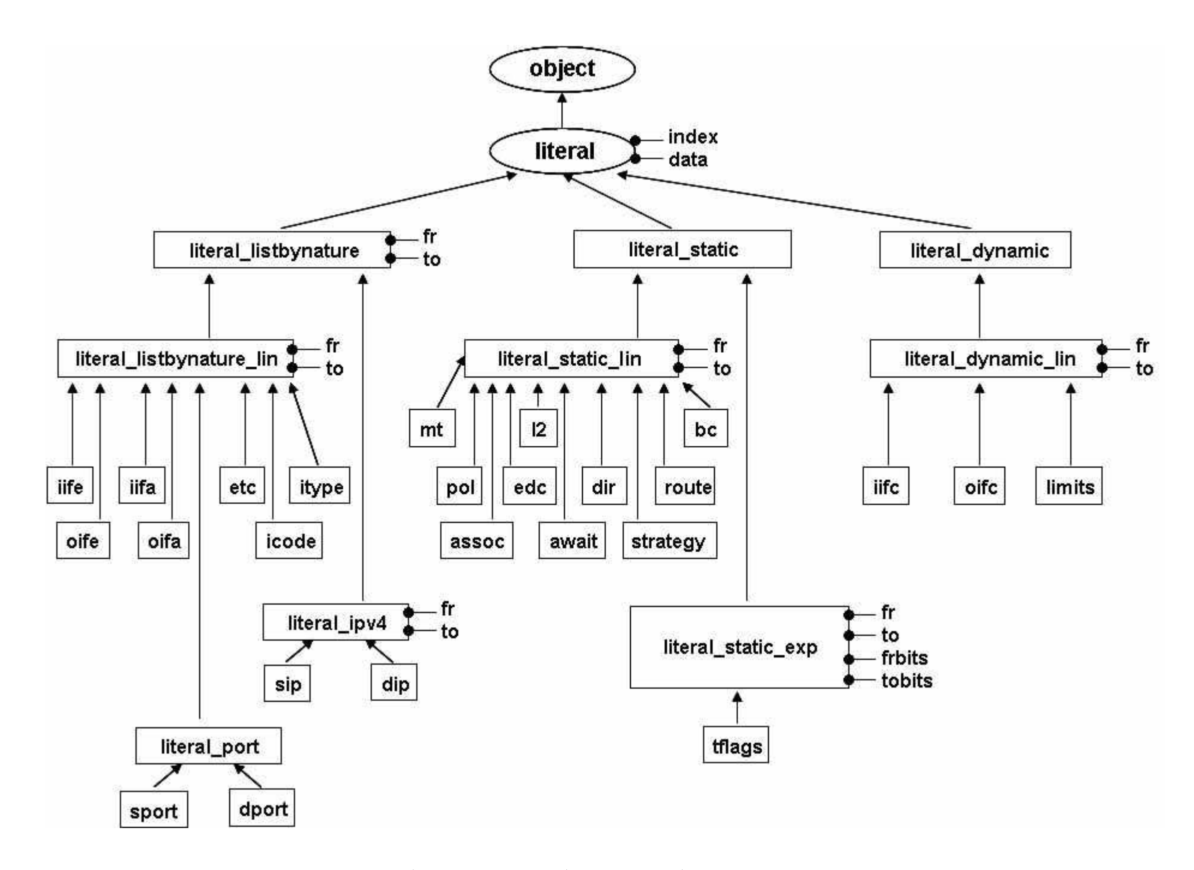


Abbildung A.1: FWA Klassen

Die Deklaration der Datenstruktur wurde in Form von Python-Klassen vorgenommen und setzt auf abstrakt definierten Datentypen aus dem Python-Modul `fwa.py` auf, welches Teil des FWA Paketes ist. Alle hier definierten und in den Vektoren benutzten Typen sind davon abgeleitet und in Abbildung A.1 dargestellt.

```
import fwa
class Path(object):
  class pol(fwa.literal_static_lin):
    val2index = {'DROP': 0, 'ACCEPT': 1, 'REJECT': 2}
  class assoc(fwa.literal_static_lin):
    val2index = {'UTR': 0, 'INV': 1, 'NEW': 2, 'EST': 3, 'REL': 4}
  # ....
  class Vector(fwa.Vector):
    def __init__(self, new_data):
      self.d = {}
      self.d['index']  = Path.progress_idx(new_data)
      self.d['proto']  = Path.proto(new_data)
      self.d['dir']    = Path.dir(new_data)
      self.d['route']  = Path.route(new_data)
      self.d['sip']    = Path.sip(new_data)
      self.d['dip']    = Path.dip(new_data)
      self.d['assoc']  = Path.assoc(new_data)
      self.d['sport']  = Path.sport(new_data)
      self.d['dport']  = Path.dport(new_data)
      self.d['tflags'] = Path.tflags(new_data)
      self.d['itype']  = Path.itype(new_data)
      self.d['icode']  = Path.icode(new_data)
      self.d['logpref']= Path.logpref(new_data)
      self.d['lpath']  = Path.linepath(new_data)
```

Listing A.2: Deklaration der Vektordatenstruktur

Im zweiten Teil wird eine Liste von Vektoren mit der Bezeichnung *path_vl* angelegt. Jeder Testvektor, wie in Listing A.3 dargestellt, beschreibt die Werte der verschiedenen Aspekte als Tupel. Einige Parameter (z.B. *'sip'* und *'dip'*) sind als Bereiche *'route'* und *'dir'* als einzelne Werte *'linepath'* als Liste und *'await'* als Hashtabelle definiert.

```
path_vl = [
    # ...
    Path.Vector( {
  'progress_idx':    7,
         'proto': ('TCP', 'TCP',),
         'route': ('UNSPEC',),
           'dir': ('FORWARD',),
           'sip': ('0.0.0.0', '255.255.255.255',),
           'dip': ('0.0.0.0', '255.255.255.255',),
         'sport': (    0, 65535,),
         'dport': (   80, 80,),
         'assoc': ('UTR','REL',),
         'await': {
              'prerouting' : {
           'nat' : [
```

```
                        ('LOG', ['log-prefix foo-bar']),
                        ('DNAT', ['to-destination 10.0.0.4:8080'])
                    ],
                    'mangle' : [
                        ('LOG', ['log-prefix foo-bar']),
                        ('MARK', ['set-mark 42']),
                    ],
                },
                'forward' : {
                    'filter' : [
                        ('ACCEPT', [] )
                    ]
                },
                'postrouting' : {
                    'nat' : [
            ('ACCEPT', [] )
                    ]
                },
        },
        'linepath': [  5, 11, 6, 3, ],
    } )
    # ...
]
```

Listing A.3: Vektorbeispiele

Zur Bedeutung der einzelnen Bezeichner:

progress_idx Nummer der Berechnung des Testvektors im FWA. Die Nummer kann mehrfach vergeben worden sein und kann nicht als eindeutige Referenz auf den Vektor verwendet werden.

proto Bereich (von, bis) der Protokolle

route Senderichtung und -ziel des Paketes aus Sicht der Firewall

dir angesprochene *Kette (Chain)* im Netfilter

sip Bereich (von, bis) der Quelladressen

dip Bereich (von, bis) der Zieladressen

assoc Bereich der Verbindungszustände

await Die kumulierten, erwarteten Reaktionen der Firewall

linepath Liste von Zeilennummern, aus denen der Vektor generiert wurde.

Dazu kommen noch protokollabhängige Angaben wie ICMP-Type bzw. -Code (itype und icode), Quell- und Zielports für TCP und UDP (sport und dport) sowie für TCP zusätzlich noch die Flags (tflags).

Die Datenstrukturen der einzelnen Vektoren werden beim Interpretieren der Datei durch die `__init__()`-Methode in der Klassendefinition erstellt. Dabei wird für alle Schlüssel, also auch die nicht explizit belegten, in den Vektoren ein Eintrag im Speicher erstellt, so dass eine einfache Überprüfung auf Existenz des Schlüssels nicht ausreicht und zusätzlich die Existenz von Daten hinter dem Schlüssel überprüft werden muss. Im negativen Fall müssen Standardwerte, die evtl. abhängig von den Daten der anderen Schlüssel sind, für diese Schlüssel in FWTStrategy vorgesehen werden. Wenn der FWA Aussagen über mehrere Protokolle trifft, so überlässt er die Belegung der protokollspezifischen Schlüssel dem Interpreter der Vektoren.

```
path_al = [
 # target , tstate , tproto , troute , cand , cstate , cproto , croute
  (   63,  'EST',  'ICMP', 'FORWARD',  344,  'EST',  'ICMP', 'FORWARD' ),
 # ...
]
```

Listing A.4: association list

In Listing A.4 wird eine Zeile aus der 'association list' dargestellt, in der der FWA bereits mögliche Kandidaten zur Herstellung einer Vorgeschichte vorschlägt. Die Liste besteht aus Tupeln mit 8 Elementen, die eine Verknüpfung zwischen dem Vektor mit der Nummer in Feld 1 (target) unter den Aspekten in den Feldern 2-4 (target state, target protocol, target route) und einem Kandidaten in Feld 5 (candidate) mit den Aspekten in den Feldern 6-8 (candidate state, candidate protocol, candidate route) herstellt.

A.3. Benutzerhandbuch

In diesem Abschnitt wird erläutert, wie eine Teststellung mit `FWTStrategy` aufgebaut werden kann. Dafür wird die Konfiguration und der Betrieb des Werkzeugs beschrieben. Für einen konkreten Einsatz werden die Ergebnisse eines Testlauf beschrieben und Hinweise gegeben, wie diese zu interpretieren sind.

Aufbau und Konfiguration einer Testumgebung

FWTStrategy wird zusammen mit dem FWA und FWTest in einer Versionsverwaltung unter `svn://seclab.kbs.cs.tu-berlin.de/fwtest` entwickelt. Für das Herunterladen der Projektquellen ist ein Subversion Client[1] notwendig. Die Software wurde für Linux entwickelt und auf mehreren aktuellen Systemen getestet. Neben den Bibliotheken, die in der Dokumentation der Werkzeuge beschrieben sind, wird Python benötigt. Es wird zumindest die Version 2.4 empfohlen. Die Dokumentation zu FWTest enthält man-pages (`fwagent.1` und `fwtest.1`), in der die Bedienung und Konfiguration beschrieben sind.

Der Betrieb von FWTStrategy wurde explizit auch in einer produktiven Umgebung vorgesehen. Für eine erste Teststellung wird jedoch eine virtuelle Umgebung empfohlen. Im Folgenden wird beispielhaft die Konfiguration einer Testumgebung unter der Virtualisierungsumgebung VMWare Server[2] beschrieben, unter der auch die Entwicklung betrieben wurde.

Für eine minimale Testumgebung werden entsprechend der Abbildung 7.2 mindestens drei virtuelle Systeme benötigt: mindestens ein System mit einem Paketfilter und zwei Systeme für die FWTest-Agenten. Die Teststeuerung selbst kann auf dem Wirtssystem ausgeführt werden. Für die Installation der virtuellen Systeme kann ein aktuelles Linux-System nach freier Wahl verwendet werden. Eine minimale Installation ohne grafische Oberfläche wird empfohlen, um Speicherplatz zu sparen. Unter den zu installierenden Werkzeugen sollte sich ein SSH-Server zur Fernbedienung, ein Netzwerksniffer und die FWTest-Agenten befinden.

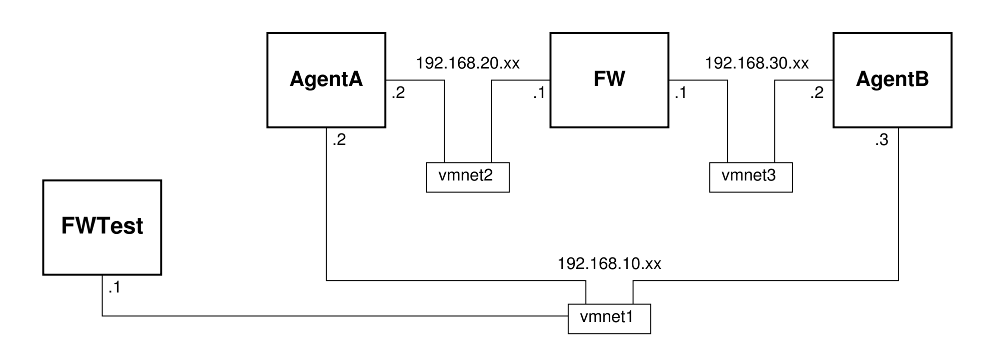


Abbildung A.2: Aufbau der Testumgebung

[1]Die Projektseite der Subversion Versionsverwaltung incl. der Dokumentation und Links zu verschiedenen Clients ist unter `http://subversion.tigris.org/` zu finden.
[2]VMWare Server ist kostenlos von dem Hersteller über `http://www.vmware.com/` zu beziehen.

Für die Verbindung der Netzwerke zwischen den Systemen werden drei Netzwerkbereiche verwendet, die sich möglichst nicht mit den Netzwerken der zu testenden Konfigurationen überschneiden sollten. Die Beispielskripte für FWTest sind auf den privaten Bereich 10.xx.0.0/16 des Klasse B Netzwerkbereiches voreingestellt. Deswegen werden für die Netzwerke zwischen den Systemen Adressen aus den privaten Bereichen 192.0.yy.0/24 empfohlen. Bei der Einrichtung der VMWare Server Umgebung werden drei virtuelle Switches konfiguriert, die in Tabelle A.4 gelistet sind.

Switch	Betriebsart	Adresse	Netzmaske
vmnet1	Host-Only	192.168.10.1	255.255.255.0
vmnet2	None	–	–
vmnet3	None	–	–

Tabelle A.4: Konfiguration der virtuellen Switche

Die virtuellen Systeme werden jeweils mit zwei Netzwerkadaptern ausgestattet, die wie in Tabelle A.5 angegeben mit den virtuellen Switches zu verknüpfen sind. Die Adressen auf den virtuellen Systemen müssen manuell konfiguriert werden. Zusätzlich muss auf dem Firewallsystem die Weiterleitung von Paketen aktiviert werden, was mit `echo 1 > /proc/sys/net/ipv4/ip_forward` eingestellt werden kann.

	Adapter 0		Adapter 1	
	IP-Adresse	Switch	IP-Adresse	Switch
Agent A	192.168.20.2	vmnet2	192.168.10.2	vmnet1
Agent B	192.168.30.2	vmnet3	192.168.10.3	vmnet1
Firewall	192.168.20.1	vmnet2	192.168.30.1	vmnet3

Tabelle A.5: Netzwerkeinstellungen der virtuellen Systeme

Für die korrekte Ausführung müssen zuerst die Netzwerkumgebungen in den Agenten definiert werden. Diese Definitionen befinden sich in einer Konfigurationsdatei unter `/etc/fwtest/fwtest.conf`. Der Abschnitt *[environment]* in der `fwtest.conf` setzt die Grob- und Feinfilter und hat zwei Angaben zur Netzwerkumgebung, wovon eine von den für Agent A und die andere für Agent B zuständig ist. Die Syntax sieht wie folgt aus:

```
[environment]
env_tn_A = [ ('10.1.0.0', '10.1.255.255'), ]
env_tn_B = [ ('10.130.0.0', '10.130.255.255'), ]
```

Bevor FWTStrategy aufgerufen wird, müssen die Agenten auf beiden Systemen gestartet werden. Danach kann FWTStrategy mit Hilfe `fwtest` gestartet werden. Ein erforderlicher Aufruf sieht wie folgt aus, wobei das gleich nach der Option *-o* gegebene Parameter eine durch `fwa` erzeugte Testvektoren-Datei ist:

```
fwtest -a 192.168.10.2 -b 192.168.10.3:1500 -f FWTStrategy.py -o udp.vec
```

Ausgaben und deren Bedeutungen

Nach der unterschiedlichen Debuglevel-Einstellung ist es möglich eine einfache oder detaillierte Ausgabe zu erhalten. Eine Beispielausgabe sieht wie folgt aus:

Wie im Abschnitt 7.1 erwähnt wurde, besteht jeder Testfall aus zwei Testpunkten (Zeile 4 und Zeile 20). In diesem Beispiel wird der Vektor 10 getestet, wobei ein ICMP-Paket von A-Seite zu B-Seite mit REL-Zustand gesendet werden soll (Zeile 14 und Zeile 23). Um einem REL-Zustand erreichen zu können wird zuerst eine Vorgeschichte mit den Zuständen NEW und EST erstellt (Zeilen 5, 8, 21 und 22). Die Zeilen 6, 9 und 15 sind Referenzen zu Regeln in der Firewallkonfiguration. Die ausführliche Information des gesendeten Pakets befindet sich zwischen den Zeilen 11 bis 13, wobei die Zeile 12 enthält, was für ein Paket der Empfänger erhalten hat.

Am Ende wird eine statistische Information ausgegeben (Zeilen 26–51). In diesem Beispiel wurden 348 Testfälle erzeugt. Davon wurden 248 Testfälle nicht getestet, wobei die Begründungen in den Zeilen davon aufgeschlüsselt sind. Aus den 79 Testfällen wurden schließlich 201 testbare Vorgeschichten abgeleitet. 388 Testfälle zeigten eine erwartete Reaktion und 14 Testfälle eine unerwartete.

```
Execute vector list from file udp.vec

testpath of vectors 6, 16, 10:
+ testpoint 1
  + [ OK ] vector:    6   [ A2B, UDP, NEW ] - ACCEPT'ed as expected
    Linepath to rule: 21
    + 10.1.0.1->10.130.0.1, port: 67 -> 67 - [OK]
  + [ OK ] vector:   16   [ B2A, UDP, EST ] - ACCEPT'ed as expected
    Linepath to rule: 21
    + 10.130.0.1->10.1.0.1, port: 67 -> 67 - [OK]
      Sent Packet: 'IP / UDP 10.1.0.1:bootps > 10.130.0.1:bootps / Raw'
      Receiver side: UDP packet
      Modified Fields: <IP: [TTL], UDP: []>
  + [ OK ] vector:   10   [ A2B, ICMP, REL ] - DROP'ed as expected
    Linepath to rule: 21, 18
    + 10.1.0.1->10.130.0.1, iType/iCode: 3/0    - [OK]
    + 10.1.0.1->10.130.0.1, iType/iCode: 3/15   - [OK]
    + 10.1.0.1->10.130.0.1, iType/iCode: 12/0   - [OK]
    + 10.1.0.1->10.130.0.1, iType/iCode: 12/1   - [OK]
+ testpoint 2
  + [ OK ] vector:    6   [ A2B, UDP, NEW ] - ACCEPT'ed as expected
  + [ OK ] vector:   16   [ B2A, UDP, EST ] - ACCEPT'ed as expected
  + [ OK ] vector:   10   [ A2B, ICMP, REL ] - DROP'ed as expected
...
...
vectors: 296
testcases:
  skipped:  78  by NOT IMPLEMENTED - untracked (UTR) association
  skipped:  14  by NOT IMPLEMENTED - UDP related
  skipped:  50  by NOT IMPLEMENTED - TCP related
  skipped:  15  by testcase already tested as prehistory
  skipped:  11  by itypes not testable in state NEW
  skipped:   8  by itypes not testable in state REL
  skipped:  32  by invalid TCP flags
  skipped:   7  by can't establish conformant packet flow from testpath
  skipped:   6  by no usable path found
  skipped:  40  by no prehistory - can't test ESTABLISHED without prehistory
  skipped:   2  by no prehistory - can't test RELATED without prehistory
           248  TOTAL skipped
            79  In Test
           342  TOTAL testcases
 testpath:
  skipped: 168  by no usable prehistory after intersection
  skipped: 534  by invalid TCP flags
  skipped: 366  by can't establish conformant packet flow from testpath
  skipped: 156  by itypes not testable in state NEW
          1224  TOTAL skipped
           201  In Test
testpoints:
    tested: 388  OK
    tested:  14  FAILED
```

Listing A.5: Ausgabe von FWTStrategy

Quellenverzeichnis

Das Quellenverzeichnis ist in zwei Teile aufgespalten. Unter den **primären Quellen** befinden sich alle publizierten Werke. Dazu werden auch Handbücher und technische Standards hinzugefügt, die teilweise online abgerufen werden können. Die referenzierten Werkzeuge und Projekte im Internet werden unter **Projektseiten** geführt. Die Literaturangaben sind alphabetisch nach den Namen der Autoren bzw. bei mehreren Autoren nach dem ersten Autor sortiert. Webseiten werden mit einem eigenem Schlüssel aufgeführt.

Primäre Quellen

[ASH03] Al-Shaer, Ehab S., und Hazem H. Hamed. „Firewall policy advisor for anomaly detection and rule editing“. In: *Proc. IEEE/IFIP 8th Int. Symp. Integrated Network Management (IM 2003)*. 2003. pp. 17–30.

[ASH04] Al-Shaer, Ehab S., und Hazem H. Hamed. „Discovery of policy anomalies in distributed firewalls“. In: *INFOCOM 2004*. Bd. 4. 2004. ISBN 0-7803-8355-9. pp. 2605–2616.

[Ayu06] Ayuso, Pablo Neira. „Netfilter's connection tracking system“. In: *;LOGIN:* Vol. 31.No. 3 (Juni 2006). pp. 34–39.

[Bär94] Bär, Udo. *OSI-Konformitätstests: Validierung und qualitative Bewertung*. VDI Reihe 10 Nr. 270. VDI-Verlag GmbH Düsseldorf, 1994.

[Bar99] Bartal, Yair, u. a. „Firmato: A Novel Firewall Management Toolkit“. In: *Symposium on Security and Privacy, IEEE* 00 (1999). ISSN 1540-7993. DOI: 10.1109/SECPRI.1999.766714.

[BCD04] Bastien, Greg, Earl Carter und Christian Degu. *CCSP Cisco Secure PIX Firewall Advanced Exam Certification Guide*. 2. Aufl. Cisco Press, 2004. ISBN 1-58720-123-2.

[BG92] Baumgarten, Bernd, und Alfred Giessler. *OSI-Testmethodik und TT-CN*. Berichte der Gesellschaft für Mathematik und Datenverarbeitung Nr. 202. Oldenbourg Verlag München, Wien, 1992. ISBN 3-486-22410-0.

[BIFABA03] Bibliographisches Institut & F. A. Brockhaus AG, Mannheim. *Brockhaus Naturwissenschaft und Technik*. Online-Edition. Bibliographisches Institut & F. A. Brockhaus AG, Mannheim, 2003.

[BS06] Von Bidder-Senn, Diana. Specification-based Firewall Testing. 533. Technischer Report. Eidgenössische Technische Hochschule Zürich, Information Security, 2006.

[BSI02] Bundesamt für Sicherheit in der Informationstechnik. *BSI-Leitfaden zur Einführung von Intrusion-Detection-Systemen. Version: 1.0.* 31. Okt. 2002. URL: `http://www.bsi.bund.de/literat/studien/ids02/index.htm` (besucht am 23. 07. 2007).

[BSI06] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grunschutzkataloge – M2.338 Erstellung von zielgruppengerechten IT-Sicherheitsrichtlinien.* 2006. URL: `http://www.bsi.de/gshb/deutsch/m/m02338.htm`.

[Buc96] Buchanan, Robert W. *Art of Testing Network Systems.* New York: John Wiley and Sons Inc., 1996. ISBN 0471132233.

[CF02] Conoboy, Brendan, und Erik Fichtner. *IP Filter Based Firewalls HOWTO.* Dez. 2002. URL: `http://www.obfuscation.org/ipf/` (besucht am 26. 06. 2007).

[CPS03] Check Point Software Technologies. *Advanced Technical Reference Guide NG with Application Intelligence.* CPTS-DOC-ATRG-01-S-NGAppInt. Ramat Gan, Israel 2003.

[DH04] Du, Yong, und Daniel Hoffman. „PBit – A Pattern-Based Testing Framework for iptables“. In: *CNSR '04: Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR'04).* Washington, DC, USA: IEEE Computer Society, 2004. ISBN 0-7695-2096-0. DOI: `10.1109/DNSR.2004.1344718`. pp. 107–112.

[EZ01] Eronen, Pasi, und Jukka Zitting. „An expert system for analyzing firewall rules“. In: *Proceedings of the 6th Nordic Workshop on Secure IT Systems.* Technical Report IMM-TR-2001-14. Copenhagen, Denmark: Technical University of Denmark, 2001. pp. 100–107. URL: `http://citeseer.ist.psu.edu/eronen01expert.html`.

[Gil02] Gill, Stephen. Maximizing Firewall Availability. Techniques on Improving Resilience to Session Table DoS Attacks. Technischer Report. 2002.

[GL05a] Gouda, Mohamed G., und Alex X. Liu. „A Model of Stateful Firewalls and Its Properties“. In: *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks.* Washington, DC, USA: IEEE Computer Society, 2005. ISBN 0-7695-2282-3. DOI: `10.1109/DSN.2005.9`. pp. 128–137.

[GL05b] Gouda, Mohamed G., und Alex X. Liu. „Complete Redundancy Detection in Firewalls“. In: *Data and Applications Security XIX*. Bd. LNCS 3654/2005. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-28138-2. DOI: `10.1007/11535706`. pp. 193–206.

[GS98] Goldsmith, David, und Michael Schiffman. *Firewalking*. Okt. 1998. URL: `http://www.packetfactory.net/projects/firewalk/firewalk-final.pdf` (besucht am 23. 07. 2007).

[Gut97] Guttman, J. D. „Filtering postures. local enforcement for global policies“. In: *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 1997. pp. 120–129.

[Hae97] Haeni, Reto E. Firewall Penetration Testing. Techn. Ber. Washington, DC, USA: The George Washington University Cyberspace Policy Institute, 1997. URL: `http://citeseer.ist.psu.edu/haeni97firewall.html`.

[HPS03] Hoffman, Daniel, Durga Prabhakar und Paul Strooper. „Testing iptables“. In: *CASCON '03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*. Toronto, Ontario, Canada: IBM Press, 2003. pp. 80–91.

[HY05] Hoffman, Daniel, und Kevin Yoo. „Blowtorch: a framework for firewall test automation“. In: *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. New York, NY, USA: ACM Press, 2005. ISBN 1-59593-993-4. DOI: `10.1145/1101908.1101925`. pp. 96–103.

[IEE90] Institute of Electrical and Electronics Engineers. *Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology*. New York, NY, USA 1990. DOI: `10.1109/MS.1989.10025`.

[IF02] Ingham, Kenneth, und Stephanie Forrest. A History and Survey of Network Firewalls. TR-CS-2002-37. Techn. Ber. University of New Mexico, Computer Science Department, 2002.

[IHAS05] Ibrahim, Adel El-Atawy, K. Hamed und H. Ehab Al-Shaer. „Policy segmentation for intelligent firewall testing“. In: *Secure Network Protocols, 2005. (NPSec). 1st IEEE ICNP Workshop on*. 2005. ISBN 0-7803-9427-5. DOI: `10.1109/NPSEC.2005.1532056`. pp. 67–72.

[ISO7498-1] International Organisation for Standardization, International Electrotechnical Commission. *International Standard ISO/IEC 7498-1:1994. Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. 1994.

[ISO9126] International Organization for Standardization, International Electrotechnical Commission. *International Standard ISO/IEC 9126. Information Technology – Software product evaluation – Quality characteristics and guidelines for their use.* 1991.

[JW01] Jürjens, Jan, und Guido Wimmel. „Specification-Based Testing of Firewalls“. In: *PSI '02: Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics.* Bd. LNCS 2244/2001. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer Verlag, 2001. ISBN 3-540-43075-X. pp. 308–316.

[Lig02] Liggesmeyer, Peter. *Software-Qualität. Testen, Analysieren und Verifizieren von Software.* Spektrum Akademischer Verlag, 2002. ISBN 978-3827411181.

[LLB02] Lidl, Kurt J., Deborah G. Lidl und Paul R. Borman. „Flexible Packet Filtering: Providing a Rich Toolbox“. In: *BSDCon.* San Francisco, USA 2002. URL: `http://www.pix.net/software/ipfw/ipfw.pdf`.

[MK05] Marmorstein, Robert, und Phil Kearns. „ITVal - A Tool for Automated IPtables Firewall Analysis“. In: *USENIX Annual Technical Conference.* Anaheim, CA, USA 2005. pp. 71–81. URL: `https://www.usenix.org/events/usenix05/tech/freenix/full_papers/marmorstein/marmorstein_html/`.

[MWZ00] Mayer, Alain, Avishai Wool und Elisha Ziskind. „Fang: A Firewall Analysis Engine“. In: *Symposium on Security and Privacy, IEEE* 00 (2000). ISSN 1540-7993. DOI: `10.1109/SECPRI.2000.848455`.

[MWZ05] Mayer, Alain, Avishai Wool und Elisha Ziskind. „Offline firewall analysis“. In: *International Journal of Information Security* 5.3 (Juni 2005). pp. 125–144. ISSN 1615-5262. DOI: `10.1007/s10207-005-0074-z`.

[Pru04] Prunoiu, Florin. *Cisco PIX Firewall - Practical Guide.* 10. Aug. 2004. URL: `http://www.enterastream.com/whitepapers/cisco/pix/pix-practical-guide.html` (besucht am 23. 07. 2007).

[Ran05] Ranum, Marcus J. *What is „Deep Inspection“?* The speaker break room, Interop (Mandalay Bay Hotel) and McCarran Airport Gate 33. 06. Mai 2005. URL: `http://www.ranum.com/security/computer_security/editorials/deepinspect/` (besucht am 23. 07. 2007).

[RFC792] Postel, Jon. *RFC792 - Internet Control Message Protocol.* 1981. URL: `ftp://ftp.rfc-editor.org/in-notes/rfc792.txt` (besucht am 26. 06. 2007).

[RFC950] Mogul, J., und Jon Postel. *RFC950 - Internet Standard Subnetting Procedure.* 1985. URL: `http://www.rfc-editor.org/rfc/rfc950.txt` (besucht am 26. 06. 2007).

[RFC1108] Kent, Stephen. *RFC1108 - U.S. Department of Defense, Security Options for the Internet Protocol*. 1991. URL: http://www.rfc-editor.org/rfc/rfc1108.txt (besucht am 26. 06. 2007).

[RFC1122] Internet Engineering Task Force, Network Working Group. *RFC1122 - Requirements for Internet Hosts – Communication Layers*. 1989. URL: http://www.rfc-editor.org/rfc/rfc1122.txt (besucht am 26. 06. 2007).

[RFC1256] Internet Engineering Task Force, Router Discovery Working Group. *RFC1256 - ICMP Router Discovery Messages*. 1991. URL: http://www.rfc-editor.org/rfc/rfc1256.txt (besucht am 26. 06. 2007).

[RFC1812] Internet Engineering Task Force, Network Working Group. *RFC1812 - Requirements for IP Version 4 Routers*. 1995. URL: http://www.rfc-editor.org/rfc/rfc1812.txt (besucht am 26. 06. 2007).

[RFC2002] Perkins, Charles. *RFC2002 - IP Mobility Support*. 1996. URL: http://www.rfc-editor.org/rfc/rfc2002.txt (besucht am 26. 06. 2007).

[RFC2663] Srisuresh, Pyda, und Matt Holdrege. *RFC2663 - IP Network Address Translator (NAT) Terminology and Considerations*. 1999. URL: http://www.rfc-editor.org/rfc/rfc2663.txt (besucht am 26. 06. 2007).

[RFC3022] Srisuresh, Pyda, und Kjeld Borch Egevang. *RFC3022 - Traditional IP Network Address Translator (Traditional NAT)*. 2001. URL: http://www.rfc-editor.org/rfc/rfc3022.txt (besucht am 26. 06. 2007).

[RFC3027] Srisuresh, Pyda, und Matt Holdrege. *RFC3027 - Protocol Complications with the IP Network Address Translator*. 2001. URL: http://www.rfc-editor.org/rfc/rfc3027.txt (besucht am 26. 06. 2007).

[RFC3489] Rosenberg, Jonathan, u. a. *RFC3489 - STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. 2003. URL: http://www.rfc-editor.org/rfc/rfc3489.txt (besucht am 26. 06. 2007).

[Roo00] Van Rooij, Guido. „Real Stateful TCP Packet Filtering in Ipfilter“. In: 2nd International SANE Conference. 2000.

[Roy87] Royce, Winston W. „Managing the development of large software systems: concepts and techniques“. In: *ICSE '87: Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987. ISBN 0-89791-216-0. pp. 328–338.

[RW02] Russell, Rusty, und Harald Welte. *Linux netfilter Hacking HOWTO*. Juli 2002. URL: http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html (besucht am 26. 06. 2007).

[Sch03] Schäfer, Günter. *Netzsicherheit - Algorithmische Grundlagen und Protokolle*. dpunkt.verlag, 2003. ISBN ISBN 3-89864-212-7.

[Sch97] Schuba, Christoph L., u. a. „Analysis of a Denial of Service Attack on TCP“. In: *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society. IEEE Computer Society Press, 1997. pp. 208–223. URL: https://www.cerias.purdue.edu/techreports-ssl/public/97-06.ps.

[Spe06] Spenneberg, Ralf. *Linux-Firewalls mit iptables & Co.* München/Germany: Addison-Wesley Verlag, 2006.

[Vig97] Vigna, Giovanni. „A Formal Model for Firewall Testing“. unpublished. 1997. URL: http://citeseer.ist.psu.edu/279361.html (besucht am 26. 06. 2007).

[VP05] Verma, Pavan, und Atul Prakash. „FACE: A Firewall Analysis and Configuration Engine“. In: *SAINT '05: Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT'05)*. Washington, DC, USA: IEEE Computer Society, 2005. ISBN 0-7695-2262-9. DOI: 10.1109/SAINT.2005.28. pp. 74–81.

[Wel07] Welte, Harald. „netfilter/iptables in embedded devices“. Persönliche Email. <laforge@gnumonks.org>. 07. Juli 2007.

[Woo01] Wool, Avishai. „Architecting the Lumeta Firewall Analyzer“. In: *Proceedings of 10th Usenix Security Symposium*. Washington, DC, USA: Usenix Association, 2001. pp. 85–97.

[Woo04] Wool, Avishai. „A Quantitative Study of Firewall Configuration Errors“. In: *Computer* 37.6 (2004). pp. 62–67. ISSN 0018-9162. DOI: 10.1109/MC.2004.2.

[X290] International Telecommunication Union - Telecommunication Standardization Sector. *X.290 - OSI Conformance Testing Methodology and Framework*. 1995.

[Yua06] Yuan, Lihua, u. a. „FIREMAN: A Toolkit for FIREwall Modeling and ANalysis“. In: *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*. Washington, DC, USA: IEEE Computer Society, 2006. ISBN 0-7695-2574-1. DOI: 10.1109/SP.2006.16. pp. 199–213.

[ZCC00] Zwicky, Elizabeth D., Simon Cooper und D. Brent Chapman. *Building Internet Firewalls*. 2. Aufl. O'Reilly Media, 2000.

Projektseiten

[AFA] AlgoSec Inc. *AlgoSec Firewall Analyzer*. URL: http://www.algosec.com/Products/FA/ (besucht am 26. 06. 2007).

[Amap] Hauser, Van, und DJ RevMoon. *Amap.* URL: http://www.thc.org/thc-amap/ (besucht am 26. 06. 2007).

[Firewalk] Schiffman, Mike D. *Firewalk.* URL: http://www.packetfactory.net/projects/firewalk/ (besucht am 06. 06. 2007).

[FPA] Al-Shaer, Ehab, und Hazem Hamed. *Firewall Policy Advisor.* URL: http://www.mnlab.cs.depaul.edu/projects/FPA/index.htm (besucht am 26. 06. 2007).

[FTester] Barisani, Andrea. *FTester.* URL: http://dev.inversepath.com/trac/ftester (besucht am 26. 06. 2007).

[fwtest] Strasser, Beat, und Gerhard Zaugg. *fwtest (ETH).* URL: http://www.infsec.ethz.ch/education/projects/archive (besucht am 26. 06. 2007).

[Hping] Sanfilippo, Salvatore. *hping.* URL: http://www.hping.org/ (besucht am 26. 06. 2007).

[IPF] Reed, Darren, u. a. *IPFilter (IPF).* URL: http://coombs.anu.edu.au/~avalon/ip-filter.html (besucht am 26. 06. 2007).

[IPFW] *Handbuch der FreeBSD IP FireWall (IPFW).* URL: http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-ipfw.html (besucht am 26. 06. 2007).

[Netfilter] Netfilter Core Team, u. a. *netfilter.org.* URL: http://www.netfilter.org (besucht am 26. 06. 2007).

[Nmap] Fyodor, u. a. *Nmap.* URL: http://www.insecure.org/nmap/index.html (besucht am 26. 06. 2007).

[PF] Hartmeier, Daniel, u. a. *OpenBSD Packet Filter (PF).* URL: http://www.benzedrine.cx/pf.html (besucht am 26. 06. 2007).

[Scapy] Biondi, Philippe. *Scapy.* URL: http://www.secdev.org/projects/scapy/ (besucht am 26. 06. 2007).

[Wireshark] Combs, Gerald, u. a. *Wireshark.* URL: http://www.wireshark.org (besucht am 26. 06. 2007).

Erklärung

Die selbstständige und eigenhändige Anfertigung versichere ich an Eides statt.

Ort, Datum *Unterschrift*